Task/Subtask IV01.7
CDRL Sequence A011–002F
8 March 1995

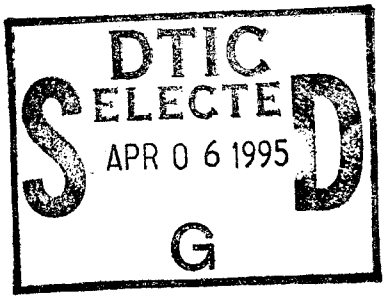
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

AF/STARS Demonstration Project Experience Report Version 2.1 (Final)

Contract No. F19628–93–C–0129
Task IV01 – Megaprogramming Demonstration Projects

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731–2116



Prepared Jointly by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

and

ESC/SRSX
130 West Paine Steet
Peterson AFB, CO 80914–2320

Cleared for Public Release, Distribution is Unlimited

Task/Subtask IV01.7
CDRL Sequence A011–002F
8 March 1995

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

AF/STARS Demonstration Project Experience Report Version 2.1 (Final)

Contract No. F19628–93–C–0129
Task IV01 – Megaprogramming Demonstration Projects

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731–2116

Prepared Jointly by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

and

ESC/SRSX
130 West Paine Steet
Peterson AFB, CO 80914–2320

Space Command and Control Architectural Infrastructure (SCAI)

Air Force / STARS Demonstration Project

Experience Report

Version 2.1 - 8 March 1995

Prepared Jointly by:

Loral Federal Systems - Gaithersburg
700 North Frederick Avenue
Gaithersburg, MD 20879

and

ESC/SRSX
130 W. Paine Street
Peterson AFB, CO 80914-2320

Executive Summary

Megaprogramming is alive and well in Colorado Springs.

A dramatic departure from the conventional, costly, problem ridden approach to software is being demonstrated on a project which teams Air Force Materiel Command (AFMC), Air Force Space Command (AFSPC) with the Advanced Research Projects Agency (ARPA) Software Technology for Adaptable, Reliable Systems (STARS) Program. A command and control system built using the ***megaprogramming*** product-line approach has shown that there is an alternative to software business as usual. The impressive results are shown in the figure below.

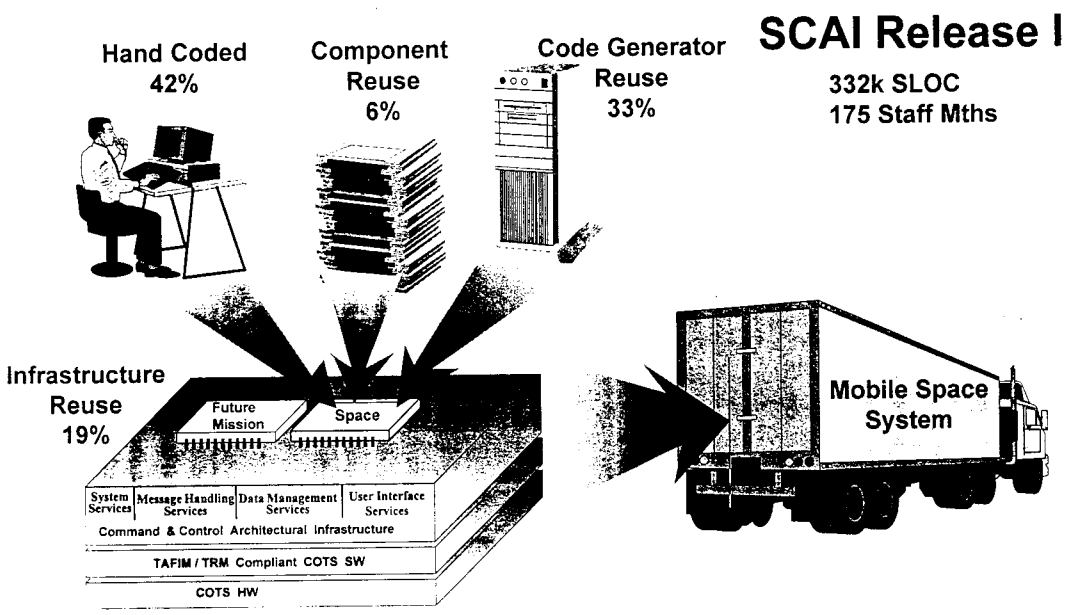


Figure 1. First Megaprogramming Release

The Space Command and Control Architecture Infrastructure (SCAI) project is now in its second and final year of development. The results shown here represent the first of three planned releases, with the final one scheduled for delivery October 1995. The total projected size of SCAI is 750K lines of code.

The first release is the result of applying the megaprogramming concept of architecture based reuse. The early results show heavy reuse coming from the infrastructure and code generation. Component reuse is expected to increase significantly as the application functionality increases.

The ultimate goal of the SCAI project is to establish a product-line basis for rapid development of future command and control capability. The SCAI project is emphasizing formal process definition and an advanced Software Engineering Environment (SEE) that will support future product-line development.

Formal process definition is being carried out in parallel with the Application Engineering and Domain Engineering activities. Process technologies being applied include IDEF modeling, ProDAT criteria-based process modeling, Entry/Task/Validation/Exit (ETVX) specifications and process driven project planning (project plan derived from the process).

This process-driven, architecture-reuse based approach demands a significant investment in automated support within the SEE. The SCAI project has built a SEE, capable of supporting the product-line, around the following major elements:

- Rational integrated tool set for OO analysis and Ada development (ROSE and APEX),
- TRW's Integrated family of code generation tools supporting the architectural infrastructure (UNAS, RICC), and
- A Process Support Environment (PSE) made up of a number of early release tools (ProDAT, PEAKS, Amadeus)

Background

The Space Command and Control Architectural Infrastructure (SCAI) Demonstration Project was selected as the Air Force STARS Demonstration Project and paired with Loral Federal Systems, Gaithersburg, MD. The project is administrated by the AFSPC Space and Warning Systems Directorate (SWSD). It is a three year program which is being conducted from October 1992 through October 1995. The partnership was chartered by a Memorandum of Agreement (MOA) signed between the Air Force and ARPA.

The MOA defines the scope of the Demonstration Project as a joint project to apply the STARS megaprogramming paradigm to the current space mission. Megaprogramming is defined by the STARS program as "process-driven, domain-specific reuse based, technology supported" software development. Personnel from Loral Federal Systems, one of the STARS prime contractors, are partners with the SCAI team. CACI, ccPE, Kaman Sciences Corp., Mantech, PRC, Rational, Robbins-Gioia, SET, and TRW perform various contractor roles on the project.

Demonstration Project Accomplishments

The SCAI project spans three years. The Preparation Phase, October `92 through October `93, was used to prototype technology, assemble the team and plan the project. The Performance Phase began in October `93 and will run for two years. This version of the Experience Report covers both the Preparation Phase and the first year of the Performance Phase.

After one year of development, Release 1 of the SCAI system has been completed (see Figure 1) verifying the potential of the megaprogramming approach. Table 1 summarizes the Demonstration Project accomplishments to date and the current status as the project begins its final year.

	Original SWSD Posture	Accomplishments since Establishing STARS Partnership	Activities in Progress (as of 1/95)
Domain Specific Reuse	• Strong architecture, based on Reusable Integrated Command Center (RICC) architectural infrastructure • Strong emphasis on Open Systems, commercial tools • Domain models underway • Commitment to Ada	• Demonstrated viability of architectural approach • Defined tailored specification standard based on Cleanroom, MIL-STD 498, and others • Completed object-based application models; specified SCAI system and first two SCAI releases • Developed and tested SCAI Release 1	• Developing SCAI Release 2; specifying Release 3 • Refining product-line architectural framework • Continuing to develop domain models
Systematic Process	• Understanding of importance of process • SWSD Software Engineering Process Group (SEPG) established • Semi-formal process definition in selected areas • Corporate Information Management (CIM) IDEF model underway	• Instituted formal approach to process definition, based on STARS/SEI collaboration • Created a product-line process architecture • Integrated OO, Cleanroom, and the Ada Process Model methods • Formally defined processes for Application Engineering (AE) • Launched a major metrics initiative • Automated staff hour and defect metrics collection	• Nearing completion of formal definition of CM process • Beginning formal definition of Domain Engineering Process • Working on second round of AE specification process • Using automated process modeling and enactment support for SCAI Release 2 and 3
Automated Support	• Commitment to Rational Ada support product-line • Commitment to Universal Network Architecture Services (UNAS), and RICC for Architectural Infrastructure	• Integrated a state-of-the-art open systems SEE: IBM and Sun platforms, Rational and TRW toolsets • Installed advanced process support toolset; encoded and began automated enactment of SCAI Release 2 and 3 processes • Instituted automated tracking capability for problems, action items, etc. • Began use of Rational SoDA for automated document production	• Enhancing the functionality and integration of the process tools • Applying Amadeus to automate collection of SEE usage metrics • Extending process automation across geographical locations

TABLE 1. Air Force Demonstration Project - Megaprogramming Progress and Status

Preparation Phase (10/92 - 10/93)

The following paragraphs present highlights of the Preparation Phase experience.

General: The Plan/Enact/Learn paradigm, elaborated in the STARS Conceptual Framework For Reuse Processes (CFRP), applies to all major project activities, including the formulation of the project's approaches and processes as well as the development of the domain and application products. Since an organization seeking to transition to megaprogramming can anticipate a significant amount of technology transition and integration, it should consciously plan an incremental build up of its approach.

Domain Engineering/Application Engineering: Architectural layering in the SWSD domain provides good guidance for forming the functional organizations needed to support the product-line.

Process: The SCAI team has elaborated the STARS Two Lifecycle Model to show very close interaction between Domain Engineering and Application Engineering, in order to constantly validate the domain models against real applications in the domain. The models must be developed iteratively to avoid the creation of complete but invalidated models. In fact, the team has developed a working hypothesis that application level models, to a large extent, should be views of domain level models.

Software Engineering Environment (SEE): Acquisition and integration for the SEE should be planned and developed in step with the incremental development of the process.

Metrics: Metrics definitions can be used to help the project focus on goals and success indicators early in project lifecycle. We successfully adjusted the Goal/Question/Metrics (GQM) approach to construe the questions as success indicators. Small focus groups of 2 to 5 people were an efficient way to refine goals, subgoals and success indicators to express the project's focus.

Technology Transition: How technology is transitioned between organizations is as important as the technologies that are transitioned. This project spent a lot of time becoming familiar with all of the technologies prior to being able to integrate them into an approach. This was true for developing our approach for defining project processes as well as for defining our Domain/Application Engineering approach.

Performance Phase, First Year (10/93 - 10/94)

The following paragraphs present highlights of the first year of the Performance Phase experience.

Domain Engineering/Application Engineering: The DE/AE teams have made significant strides on the SCAI Project, with many lessons learned from the preliminary domain analysis and subsequent development of the first megaprogramming release. We found that a domain architecture helps in managing engineering activities by providing a clear basis for organizing cooperating but "loosely coupled" groups of developers. We increased our appreciation of the need to integrate DE and AE processes in order to leverage the expertise of both disciplines and avoid the tendency for the two groups to drift in different technical directions.

We also gained first-hand knowledge of the synergy between domain-specific software reuse and demonstration-based architectural development. Demonstration-based software development (Ada Process Model) identifies architectural errors early in the development process before they become expensive to fix; domain-specific software reuse provides software components for assembling demonstrations that are progressively more accurate.

Process: Formal process definitions - including process models, and enactment information templates - have been developed for several SCAI processes, and to varying extents full process execution and improvement cycles have been realized. The cost of formal process definition is significant, particularly for an organization with little prior experience in conducting such activities. We found that close to 20% of the effort on the Demonstration Project went into process definition. In retrospect, the initial level of detail attempted for defining these processes was excessive - particularly given the limitations of the supporting toolset - and the project is currently exploring improvements to streamline the approach, reduce the number of tools (and associated databases), and upgrade the capabilities of the remaining tools.

Based on the experience of defining and using these processes, we believe that a project needs a technical leader responsible for process architecture and that the process should be developed iteratively.

Software Engineering Environment (SEE): According to our recent SEE survey, the Demonstration Project SEE is serving the end-users reasonably well and appears to be headed in the right direction. The main state-of-the-art technology thrust of the Demonstration Project is process and SEE support for process. The project has faced significant challenges in both, and it is clear that the incremental, iterative approach is the only way to go. There are promising results to date, and much interesting work left ahead as we work towards a SEE to support a full SWSD product-line.

Judging from our experience to date, the highest payoff path to effective SEE integration is to adopt strategic toolsets from vendors whose long-range product vision is aligned with the SWSD's product-line objectives. In our case, these vendors are Rational (Ada, OO modeling, automated documentation production) and TRW (architectural infrastructure support). Even with such toolsets, however, the product-line organization must have its own SEE engineering expertise to customize and tailor off-the-shelf tools, to implement integration at the total SEE level, and to plan and implement continual SEE improvements as the product-line needs evolve.

Metrics: During the first half of the Performance Phase the SCAI metrics team fulfilled its goals of implementing goal-directed measurement of the Demonstration Project. Effort by major activity and problem

tracking relative to the SEE were implemented. Mechanisms for further metrics collection were developed and are being tested before being implemented across the project. These include effort by software unit, tool usage, and software unit size. Three Metrics Reports were published to record our progress during the year. The major lessons learned include the need to motivate the demand for metrics at all levels of an organization. Management must demand visibility into the process and products. Feedback to developers must be provided to summarize the quality and quantity of their work and to provide incentive for improvement. A full-time metrics coordinator is required to keep the process moving forward. Additionally, the team gained experience developing a business case for new technologies and computing a return on investment. Further work in all these areas is planned.

Technology Transition: The transition of many new technologies into a project made up of the Air Force and 10 different contractors has proved to be a challenging task. The lessons learned include the need for a formal technology transition plan, updated based on experience. Avoid transitioning multiple technologies simultaneously, but if you must do this expect additional complications with technology integration. Use an iterative approach to technology transition and process improvement. Be realistic with your training plan in order to balance the need for significant amounts of training with the need to get the project going and gain some experience with the technology. And finally, pilot the new technology with actual user involvement prior to release for operational use.

Bottom Line

Our experience has shown megaprogramming, when combined with an architectural infrastructure approach, results in dramatic improvements in cutting delivery time, improving quality, and lowering cost of ownership. Faster. Better. Cheaper.

Preface

This document was developed by the Loral Federal Systems-Gaithersburg, located at 700 North Fredrick Avenue, Gaithersburg, MD 20879 and U.S. Air Force/SWSD/SMX, 130 West Paine Street, Peterson AFB, CO 80914-2320. Questions or comments should be directed to Major Clint Heintzelman at 719-554-6533 (cheintze@spacecom.af.mil) or Dr. Richard Randall at 719-554-6597 (randallr@lfs.loral.com).

Contributions to this document have been provided by the following organizations: Air Force Materiel Command/Space and Warning Systems Directorate, CACI, ccPE, Kaman Sciences Corp, Loral Federal Systems, Mantech, PRC, Rational, Robbins-Gioia, SEI, SET, and TRW.

Forward

Purpose

This Experience Report is a product of the Air Force/STARS Demonstration Project, a project intended to demonstrate megaprogramming during the development of a real system - the Space Command and Control Architectural Infrastructure (SCAI) application. The purpose of the approach is to present experience and lessons learned in the course of applying megaprogramming technologies. Specifically, the report will present the lessons learned in applying the STARS process driven, technology supported, domain-specific reuse technologies to:

(1) Further the establishment of a product-line organization within the Space and Warning Systems Directorate (SWSD).

(2) Implement an operational space control capability for the Commander In Chief (CINC) Mobile Alternate Headquarters (CMAH).

Audience

This document is targeted to a number of different audiences.

(1) Advanced Research Projects Agency (ARPA)

(2) Loral Federal Systems Group, Loral Corporation

(3) Space and Warning Systems Directorate (SWSD)

(4) Air Force Space Command

(5) Air Force Materiel Command

(6) Department of Defense Community

(7) Software Engineering Community

Assumptions

This document is written with the following assumptions about its audience:

(1) The reader has a working knowledge of software engineering practices, management, development and evolution.

(2) The reader has a working knowledge or an interest in gaining an understanding of the support elements that make up the STARS megaprogramming paradigm: process-driven development, domain-specific reuse, and Software Engineering Environments.

Organizational Changes

In January, 1994, after the conclusion of the Preparation Phase, IBM Federal Systems was acquired by Loral Corporation and became Loral Federal Systems. Most references in this report are to Loral Federal Systems, but some references to IBM Federal Systems remain for historical purposes.

In February, 1995, after the conclusion of the first year of the Performance Phase, the Space and Warning Systems Center (SWSC), formerly part of the Air Force Space Command (AFSPC), became the Space and Warning Systems Directorate (SWSD), now part of Air Force Materiel Command (AFMC). Please bear this recent organizational change in mind while reading this Experience Report, since most of the report was written prior to the change and still refers to the old names.

Document Structure

This document is organized into 7 chapters. A description of the contents of each chapter is provided in the following table (see Table 1).

Chapter	Title	Description
1	Introduction	Provides a summary of the project history, vision, mission, goals, technical approach and schedule.
2	Domain Engineering/ Application Engineering	Summarizes the project's experience in integrating STARS technologies with incumbent approaches to develop engineering processes that address the scope of both Domain and Application Engineering, and utilizing and improving that process to develop a C^2AI space application. Summarizes the project's experience in testing the applicability of expanding the C^2AI to the space domain.
3	Process Support	Summarizes the project's experience in transitioning the organization to be process-driven, developing the SCAI Process Architecture, manually enacting and making improvements to the defined process, and preparing an automated Process Support System.
4	SEE Support	Summarizes the project's experience in integrating and using advanced software tools into a Software Engineering Environment (SEE).
5	Metrics	Summarizes the project's experience in developing a metrics strategy, and implementing a goal directed metrics program.
6	Technology Transition	Summarizes the project's experiences in transitioning a number of new technologies into a project made up of the Air Force and 10 different contractors.
7	Conclusion	Final Remarks

TABLE 2. Document Organization

This is the second of three releases of this report. The first release covered the Preparation Phase activities. This report adds to the original and covers the first year of the Performance Phase. Sections 2 through 6 are broken into two portions covering each of these phases. A final report will be published in early 1996 which adds the experiences of the last year of the Performance Phase.

Table of Contents

Appendices

List of Figures

List of Tables

1.0 Introduction

The purpose of this report is to present the experience and lessons-learned in the course of applying Software Technology for Adaptable, Reliable Systems (STARS) megaprogramming technologies to the Space Command and Control Architectural Infrastructure (SCAI) Demonstration Project. The report covers experiences related to Domain Engineering/Application Engineering, Process Support, Software Engineering Environment Support, Metrics, and Technology Transition. For each of the above areas the report provides a brief description and overview, followed by plans, summary of accomplishments, lessons learned, and recommendations.

The SCAI project spans three years. The Preparation Phase, October `92 through October `93, was used to prototype technology, assemble the team and plan the project. The Performance Phase began in October `93 and will run for two years. This version of the Experience Report covers both the Preparation Phase and the first year of the Performance Phase.

1.1 Project Context

1.1.1 The STARS Program

The Advanced Research Projects Agency (ARPA) STARS Prime program was initiated in 1988 to create cooperative development work from a very broad industry experience base, reduce risk, and accelerate acceptance of changing technology. The contract was awarded to three leading defense systems integrators, Boeing Defense and Space Group, IBM Federal Systems Division (now Loral Federal Systems), and Unisys Government Systems Group.

The current STARS program is a technology development, integration and transition program to demonstrate a process-driven, domain specific, reuse-based approach to software engineering that is supported by appropriate tool and environment technology. This approach is often referred to as "megaprogramming."

Megaprogramming is a product-line (family of systems) approach to the creation and maintenance of software intensive systems. It is characterized by the reuse of software lifecycle assets within a product-line including common architecture and components. Megaprogramming also includes the definition and enactment of disciplined processes for the development of applications within the product-line and for the development and evolution of the product-line itself. [1]

A mission of the STARS program is to accelerate the transition to the megaprogramming paradigm. To accomplish this, STARS is jointly sponsoring Demonstration Projects with each of the three Services. The major objectives for each of the projects are to:

(1) Apply a megaprogramming paradigm to the development of software for an actual Department of Defense (DoD) system. The system will be produced using new approaches, and will establish the credibility of the megaprogramming approach.

(2) Make quantitative and qualitative measures of the effects of megaprogramming on the development effort and the resulting product. Produce reports documenting lessons learned in applying megaprogramming that will help others as they begin using megaprogramming and provide feedback on how tools and processes worked in practice.

(3) Transition to the Demonstration Project organization the capability to practice megaprogramming. When this demonstration is over, the Demonstration Project organization will be able to continue using megaprogramming on this project and should be able to apply megaprogramming to other projects without assistance from STARS.

1. Abstracted from a paper titled *STARS Program History 1983-1993* Version 1.1 assembled by Joel Trimble.

1.1.2 Demonstration Project Background

Despite DoD attempts to curtail soaring software development and maintenance costs, the cost of supporting mission software for United States Space Command (USSPACECOM), North American Aerospace Defense Command (NORAD) and Air Force Space Command (AFSPC) continues to rise. The AFSPC Space and Warning Systems Center (SWSC[1]) is responsible for the maintenance and evolution of mission-critical software meeting operational requirements for NORAD, USSPACECOM and AFSPC for the Command and Control (C^2) centers for the Cheyenne Mountain Air Force Station (CMAS), as depicted in Figure 1. These C^2 centers are responsible for national attack warning/assessment and space surveillance/defense/control.

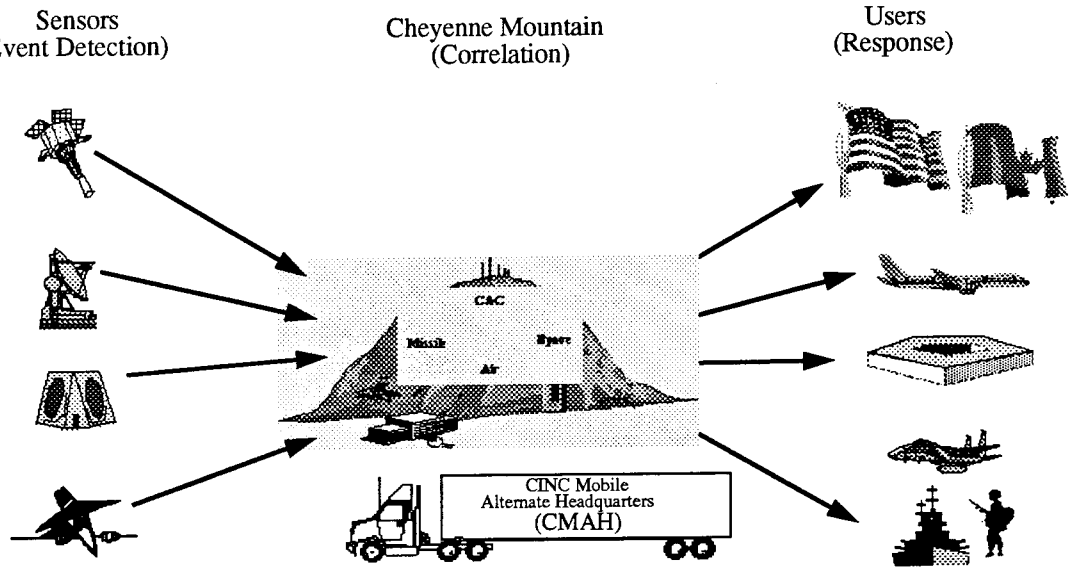


Figure 1. CMAS Systems

In addition to direct software support, the SWSC provides network software engineering, configuration management, security and technical support for communications, space and warning systems. The AFSPC SWSC has the mission to:

(1) Support combat operations by developing and maintaining mission-critical software to meet the requirements of NORAD, USSPACECOM and AFSPC command, control and intelligence centers.

(2) Ensure the operational and technical integrity of national attack warning and space control systems and provide space and warning software engineering and configuration management.

The AFSPC legacy predates the recognition of the importance of a disciplined software engineering process. Stovepipe[2] implementations have precluded reuse, diverse tools and methods have prevented resource sharing, and system-oriented organizations have inhibited the transfer of technologies. The SWSC currently maintains in excess of 12 million lines of code on 34 separate operational systems written in 27 languages - a situation which results in excessively high software maintenance costs. Accordingly, the SWSC has a priority to overhaul its software processes to take advantage of updated technologies.

In 1990 the AFSPC SWSC initiated an effort to develop a strategy to move to a common software architecture based on open systems environments as quickly as possible. While the Air Force usually procures systems through Electronic Systems Center (ESC), in some cases, on an experimental basis, the SWSC has

1. As of February, 1995, the Space and Warning Systems Center (SWSC) is now the Space and Warning Systems Directorate (SWSD) and has transitioned to Air Force Materiel Command. Refer to Section 1.1.3 for additional explanation.
2. No intent to share architecture or information except at interfaces.

attempted to perform some new development itself. As a result of one of these experiments, the SWSC, in cooperation with TRW, set about to demonstrate large scale reuse. They developed a set of architectural components, called the Command and Control Architecture Infrastructure (C^2AI), that promises to significantly reduce systems development time and cost while increasing quality.

The concept for the C^2AI was developed in the SWSC as a result of analyzing the systems that the SWSC is responsible for. Leveraging work accomplished by TRW on a production contract called Command Center Processing and Display System Replacement (CCPDS-R), the SWSC contracted TRW to advance CCPDS-R techniques and demonstrate a subset of the Cheyenne Mountain Missile Warning requirements, on a pilot program called the Reusable Integrated Command Center (RICC).

The RICC technology, developed for the SWSC by TRW under the Air Force Embedded Computer Resource Support Improvement Program (ESIP) in 1991, yielded the first set of reusable artifacts supporting a new common architectural approach including tools for generating Ada code and definition files for an application, plus the domain reusable components.

In mid 1992, the SCAI project was identified as a strong candidate, and in early 1993, the Air Force and ARPA, for the STARS Program, executed a Memorandum of Agreement (MOA) committing both sides to collaborate in addressing their mutual objectives. Loral, one of the STARS prime contractors, began working with the SWSC and its contractors to understand the SWSC domain and its processes, and to provide early coaching on relevant STARS concepts and technologies.

1.1.3 Organizational Changes

In January, 1994, after the conclusion of the Preparation Phase, IBM Federal Systems was acquired by Loral Corporation and became Loral Federal Systems. Most references in this report are to Loral Federal Systems, but some references to IBM Federal Systems remain for historical purposes.

In February, 1995, after the conclusion of the first year of the Performance Phase, the Space and Warning Systems Center (SWSC), formerly part of the Air Force Space Command (AFSPC), became the Space and Warning Systems Directorate (SWSD), now part of Air Force Materiel Command (AFMC). Please bear this recent organizational change in mind while reading this Experience Report, since most of the report was written prior to the change and still refers to the old names.

1.1.4 Summary of Demonstration Project Objectives

The SCAI project has joint goals of technology demonstration/transition and development of a deployable space capability for the Commander in Chief (CINC) Mobile Alternate Headquarters (CMAH). The technologies to be demonstrated are STARS "megaprogramming[1]" technology and RICC technology. Following the success of the SCAI project, other SWSC programs will be transitioned to use the new technologies.

SWSC Vision

The SWSC must significantly improve its ability to provide mission capable software to the war-fighters. To address this long-range objective, the SWSC's vision associated with the Demonstration Project is:

> "We are a highly productive, thriving engineering team, using megaprogramming to provide mission-effective systems in our product-line, and delighting our customers with our quality and responsiveness."

Plans and Engineering Directorate (SMX) Mission

The SMX mission during the Demonstration Project's time frame is to:

> "Formalize a ***practical megaprogramming process*** that combines the best of the STARS and RICC technologies, proving its viability by building and deploying an operational system in the space and warning domain and initiating its transition throughout the SWSC."

1. Megaprogramming is defined by the STARS program to be "process-driven, domain-specific reuse based, technology supported" software development.

Demonstration Project Goals

Table 1 captures the top level goals for the SCAI Project from a consistent view across all three STARS Demonstration Projects (i.e., for the Army and Navy projects as well as the Air Force project described in this report).

Capability	Current Baseline	SCAI Goal Product/Approach	Organizational Vision *Institutionalization*
Space Software Baseline	FORTRAN, JOVIAL, Assembler (3.5M LOC)	Reengineered in Ada using system architecture	Multiple space systems based on common models and reusable assets
Software Process (SWSC)	Defined high level processes. Repeatable at lower levels.	Partial process-driven, reuse-based capability	Managed with continuous process improvement
Software Reuse (SWSC)	Prototype architectural infrastructure with code generators (UNAS & RICC)	Domain requirements and architectural models, domain library, & reusable assets	Product-line process with managed reuse
Software Architecture (SWSC)	System-Specific/Evolution Limited	Evolution enabled	SWSC-wide architectural strategy
Software Engineering Environment (SWSC)	System-Specific	Partial enactment with integrated processes	Adaptable and open SWSC SEE strategy, integrated assets
People Skills(SWSC)	Diverse, System-Specific	Trained in SCAI technologies	Focused on Technologies for Product-Line

Table 1. STARS View of AF/STARS Demonstration Project

SCAI Product Goals: The Demonstration Project product goals, in building a real system, are:

- Develop a space mission capability suitable for operational deployment by October, 1995, and
- Develop reusable domain assets (architecture, models, components, processes, etc.) that can be applied to future product-line applications.

SCAI Approach Goals: The Demonstration Project approach goals, in demonstrating the benefits of using a practical megaprogramming approach that combines the best of the STARS and RICC technologies to build the system, are:

- Apply the RICC technology to SCAI development and demonstrate its benefit,
- Create a megaprogramming process by instituting cooperating domain engineering and application engineering (DE/AE) product line processes including those for ongoing process improvement,
- Instantiate a process-driven Software Engineering Environment (SEE) to support megaprogramming, and
- Demonstrate that the SCAI application was built "faster, better, and cheaper" using the new process.

SWSC Institutionalization Goals: During the Demonstration Project time frame the goal is to initiate the institutionalization of the megaprogramming paradigm used to develop the SCAI. The project will:

- Demonstrate widespread dissemination of technology expertise within the SWSC organization and its contractors,
- Establish a product-line organization and infrastructure, managing the evolution of multiple application systems, and
- Transition technology/assets to another organization for their own pilot application.

Table 2 lists the Demonstration Project goals presented in Section 1.1.3 and indicates how each of the four major experience areas address them. In this table, the letter P is used to designate goals that are considered primary focuses of the area; S is used to designate goals that are supported to a significant degree by the area.

Demonstration Project Goal	Experience Area			
	Domain / Application	Process Support	SEE Support	Metrics
PRODUCT GOALS				
Develop a space mission capability suitable for operational deployment by October, 1995.	**P**		**S**	
Develop reusable domain assets	**P**	**S**	**S**	**S**
APPROACH GOALS				
Apply the RICC technology to SCAI development and demonstrate its benefit.	**P**	**S**	**S**	**S**
Institute cooperating DE/AE processes including those for ongoing process improvements.	**P**	**P**		
Instantiate a process-driven SEE to support megaprogramming.		**S**	**P**	**S**
Demonstrate that the SCAI application was built "cheaper, better, and faster" using the new process		**S**	**S**	**P**
INSTITUTIONALIZATION GOALS				
Demonstrate widespread dissemination of technology expertise within the SWSC organization and its contractors				**S**
Establish a product-line organization and infrastructure, managing the evolution of multiple application systems	**S**	**S**	**S**	**S**
Transition technology/assets to another organization for their own pilot application				**S**

Table 2. Applicability of Project Goals to Key Technology Areas

1.1.5 Relationship Between Goals and Metrics

SCAI metrics will be used to provide project-wide visibility into team performance with respect to the project's objectives. Three quantifiable attributes of any software project are its cost, duration, and quality. The SCAI efforts to improve the software development process are directed at simultaneously reducing cost, shortening schedule, and improving quality rather than simply trading one attribute for another. This project will continually refine and formalize a megaprogramming product line process and assure that reusable assets are developed, identified, and packaged. In addition, specific megaprogramming technologies, such as technology supported process driven development, will be used and the impacts of their use on the project cost, schedule, and quality will be identified. Metrics will be instrumental in providing information in order to quantify software lifecycle cost, quality, and capability differences resulting from the presence or absence of STARS megaprogramming technologies. The National Aeronautics & Space Administration (NASA) Software Engineering Laboratory (SEL) Goal/Question/Metric paradigm will be used to identify specific goals, define success criteria, and specify metrics that determine success or failure compared with the criteria. The Institute for Defense Analyses (IDA), under a STARS contract, provides assistance in defining and gathering metrics data.

Long Range Objective

The Air Force's long-range objective is to realize a product-line organization, in which a wide range of its applications are developed and maintained using a megaprogramming process. The partnership with STARS on the SCAI Demonstration Project is helping the Air Force accelerate its progress toward this objective.

The SWSC is responsible for the maintenance of a wide range of space and warning C^2 applications. The SWSC's long-range intent is to manage as many systems as possible within this domain of applications using a coherent process; and an envisioned product-line organization is intended to facilitate this type of management. There are three fundamental prerequisites for a successful product-line organization:

(1) A domain manager responsible for all applications in the product-line,

(2) Sufficient commonality among the applications to allow management of the entire set as a whole, and

(3) Enabling technology to facilitate the systematic management and engineering work.

To prepare for this type of organization, the SWSC is working with STARS to mature its technological approach, using the SCAI Demonstration Project as its primary vehicle.

Figure 2 depicts the SWSC's current concept of the future product-line organization and illustrates the role of the STARS megaprogramming technology areas in working towards this organizational objective. Appendix C has further discussion of this topic.

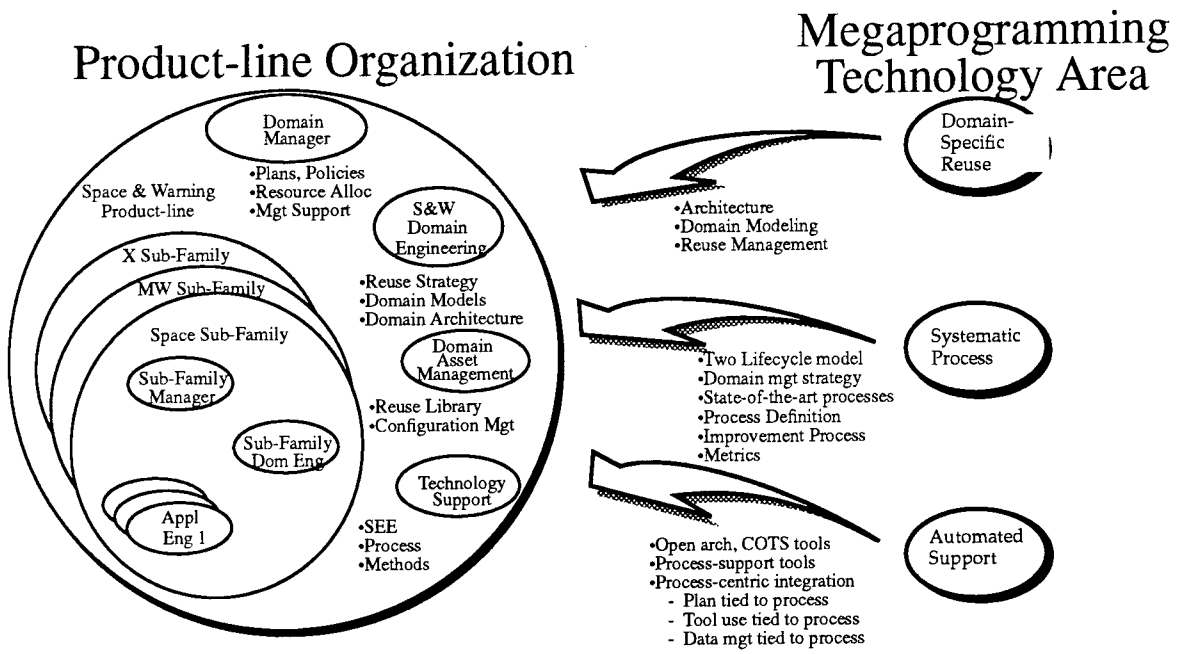


Figure 2. Megaprogramming: Enabling a Future Product-line Organization

1.2 Overview of Approach

1.2.1 Demonstration Project Schedule

The Demonstration Project is performed during the three phases described below and shown in Figure 3. The three phases are:

(1) A **Preparation Phase** where the goal is to ensure that all the technologies, assets, and training required to support use of the megaprogramming paradigm are in place.

(2) A **Performance Phase** where the goal is to apply and measure the use of megaprogramming technologies to establishing a product-line organization with the SWSC, and implementing an operational space control capability for the CMAH.

(3) A **Reflection Phase**, where the goal is to produce reports and presentations that document the significant results from the Demonstration Project, including lessons learned that will help others in adopting the megaprogramming approach. Reports will also be generated which quantify or project the benefits obtainable from the use of megaprogramming.

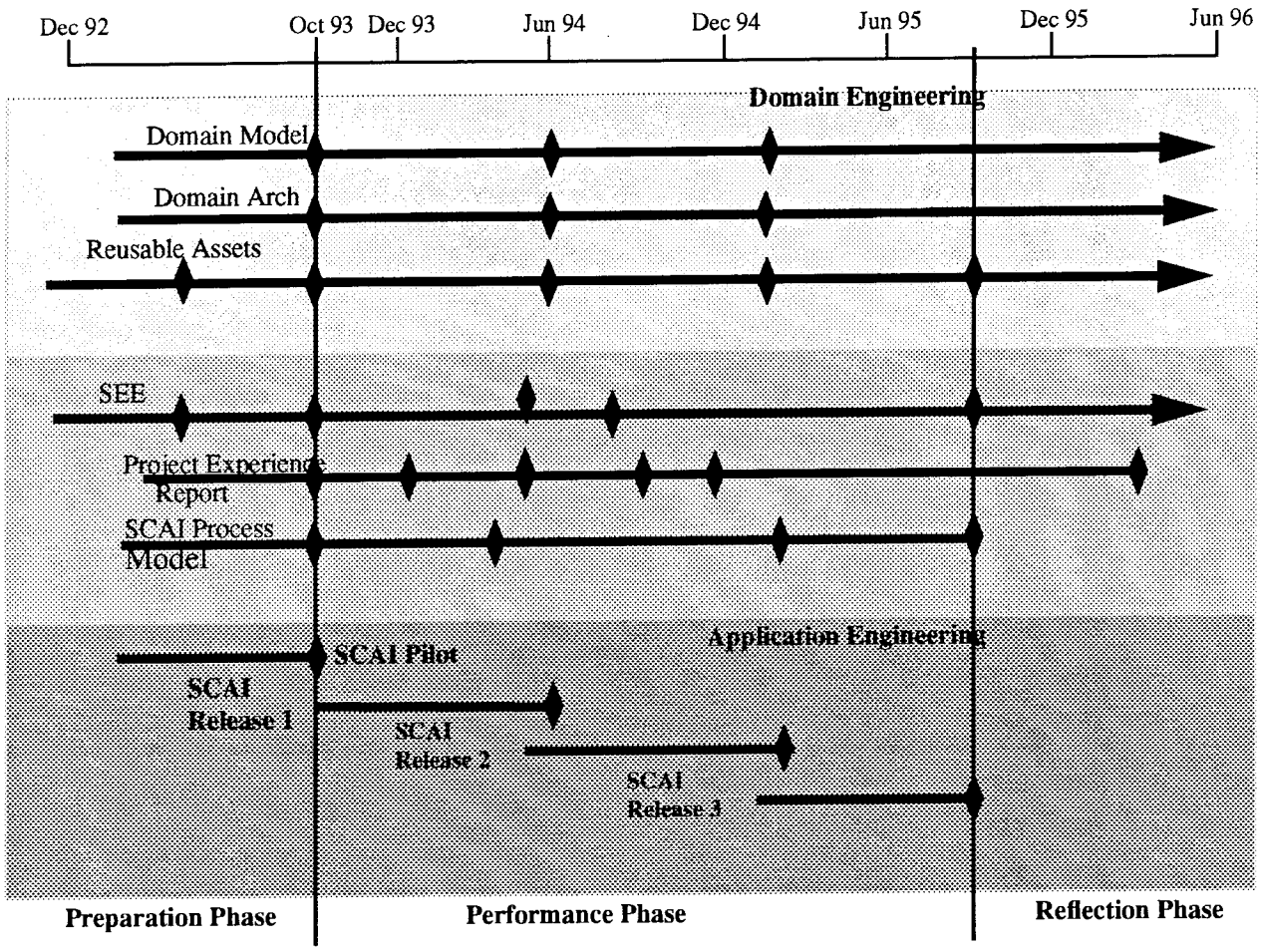


Figure 3. Project Schedule

1.2.2 Basic Strategy

The Demonstration Project effort was originally divided into three phases in accordance with the demonstration concepts document as depicted in Figure 4. (As of this writing (11/94), the project is in the Performance Phase.)

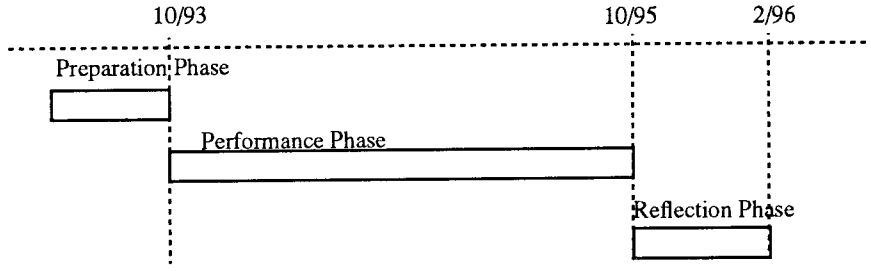


Figure 4. Demonstration Project Schedule Model

A revised model has been developed in recognition that preparation activities overlay the Performance Phase, that certain Performance Phase activities will not require support at the outset, and that early activities will disclose the need for adjustments in the approach. Figure 5 depicts the improved model.

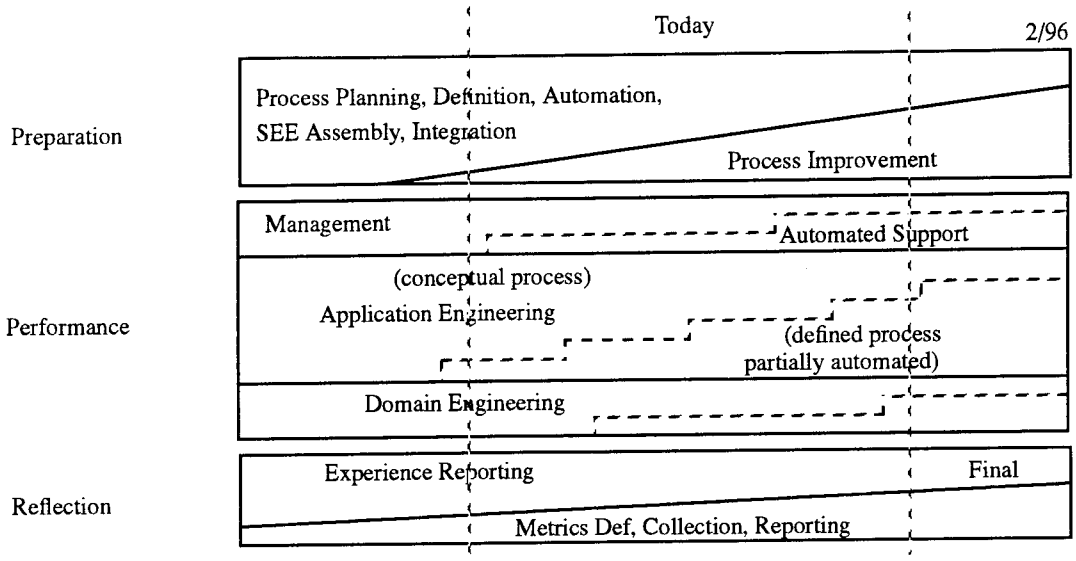


Figure 5. Revised Project Schedule Model

1.2.3 Preparation Phase

During the Preparation Phase, Loral was paired with SWSC/SMX, the Air Force organization responsible for the SCAI application, and assisted in planning and preparing for the Performance Phase. SMX, with its contractor force, strengthened its understanding of the domain, learned megaprogramming concepts, and produced several artifacts to provide a foundation for Performance Phase activities.

Major objectives of this phase were:

- Obtain funding for the Performance Phase,
- Assemble the Performance Phase team and gain experience in working together,
- Define the processes to be used during the Performance Phase as well as the requisite SEE support,
- Acquire the SEE hardware and software and perform integration sufficient to support initial Performance Phase processes,
- Train the team in using the SEE and in carrying out the processes,
- Conduct pilots to gain experience with the SEE and the processes and to refine the approach,
- Prepare key artifacts to document domain understanding, and
- Define the artifacts to be produced during the Performance Phase and determine how to adapt the Preparation Phase artifacts to support the Performance Phase.

Formulating the SCAI technical approach involved evaluating a number of state-of-the-art technologies and tools, deciding which were most applicable, and determining how best to adapt and combine them. Although this proved to be a significant challenge, by the end of the Preparation Phase the team had agreed on the overall approach and had defined a preliminary process architecture.

Table 3 identifies some of the more significant technologies and tools that were considered. All of these technologies and tools are discussed in the four major experience report sections that follow. In addition, Appendix B provides further details for those items flagged with an asterisk (*) in the table.

The table divides the technologies and tools into three categories. Incumbent SWSC Technologies and Tools are those that were either already in use at the SWSC or had been pre-selected by Air Force manage-

ment. STARS-Sponsored Technologies and Tools are those that had been developed or adopted by STARS as enabling technologies in support of megaprogramming. Jointly Developed Technologies and Tools are those that were formulated during the Preparation Phase by the AF/STARS team.

	Incumbent SWSC Technologies and Tools	STARS-Sponsored Technologies and Tools	Jointly Developed Technologies and Tools
DE/AE	CIM $IDEF_0$ Modeling * Mission Operation/Information Analysis (MOIA)* Ada Process Model* Booch Object Oriented Analysis* RICC-based Architectural Infrastructure*	Conceptual Framework for Reuse Processes (CFRP) Two Lifecycle Model for Coordinated DE/AE Processes Domain Analysis Process Model (DAPM)* Cleanroom Software Engineering Process *	
Process	Corporate Information Management (CIM) Process Initiative $IDEF_0$ Modeling *	Process Driven Management and Engineering Approach * ETVX Process Modeling	Information Organizer Templates (with SEI)*
SEE	Reusable Integrated Command Center (RICC) toolset* Rational Ada Support toolset	Software Process Management System (SPMS)* CAT/Compass* ProjectCatalyst* Process Weaver*	
Metrics		Amadeus*	Goal-based Metrics Process

Table 3. Background Technologies and Tools

1.2.4 Performance Phase

Major objectives of this phase are:

- Retain and strengthen systems engineering and integration responsibility within the Air Force, assigning major subtasks to appropriate contractors,
- Achieve a deployable system which includes generating and gaining approval for a deployment plan as well as an operational test plan, and then performing to that plan,
- Specify and evolve the megaprogramming engineering approach in a planned phased fashion so that processes and SEE support are ready in time to support the scheduled incremental development activities,
- Continuously improving the process, and
- Implement an effective technology transition plan that will ultimately result in inserting the capability within the SWSC to manage and support megaprogramming for a future product-line of applications.

1.2.5 Reflection Phase

Major objectives of this phase are:

- Evaluate effectiveness of megaprogramming,
- Gather and consolidate legacy materials, and
- Consolidate experience and lessons learned.

1.3 Experience Overview

1.3.1 Key Accomplishments

At the time of publication of this version of the Experience Report, Release 1 has been completed. Figure 6 provides a summary of the Release 1 characteristics.

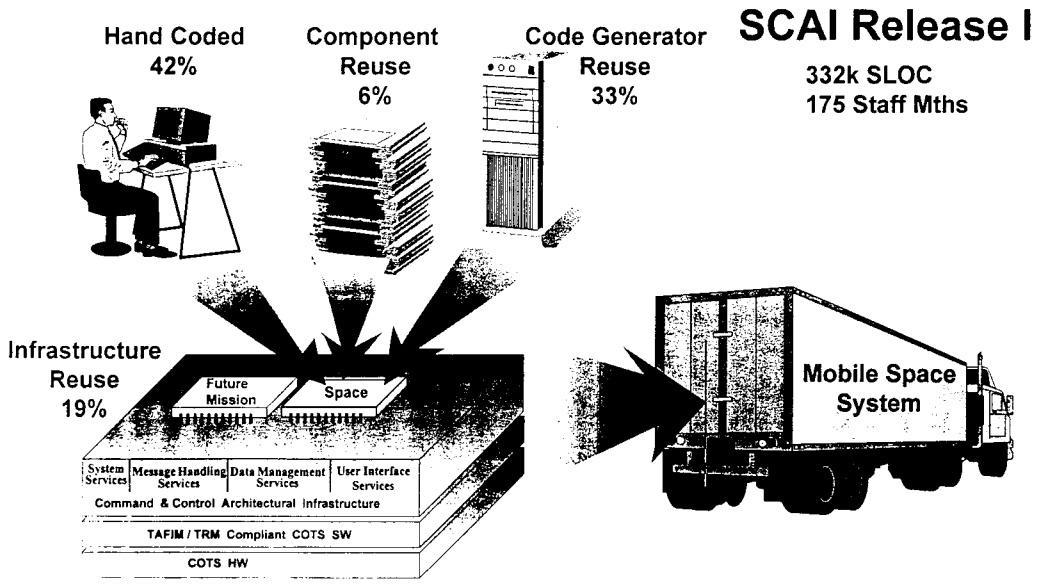


Figure 6. First Megaprogramming Release

From a technology standpoint, Table 4 illustrates some of the concrete strides that have been made through both the Preparation Phase and the Performance Phase to date. The table is divided into three sections -- one for each of the three main aspects of megaprogramming (process-driven, domain-specific reuse-based, and technology supported); the columns depict how the team is progressing in each.

	Original SWSC Posture	Accomplishments since Establishing STARS Partnership	Activities in Progress (as of 1/95)
Domain Specific Reuse	• Strong architecture, based on Reusable Integrated Command Center (RICC) architectural infrastructure • Strong emphasis on Open Systems, commercial tools • Domain models underway • Commitment to Ada	• Demonstrated viability of architectural approach • Defined tailored specification standard based on Cleanroom, MIL-STD 498, and others • Completed object-based application models; specified SCAI system and first two SCAI releases • Developed and tested SCAI Release 1	• Developing SCAI Release 2; specifying Release 3 • Refining product-line architectural framework • Continuing to develop domain models
Systematic Process	• Understanding of importance of process • SWSC Software Engineering Process Group (SEPG) established • Semi-formal process definition in selected areas • Corporate Information Management (CIM) IDEF model underway	• Instituted formal approach to process definition, based on STARS/SEI collaboration • Created a product-line process architecture • Integrated OO, Cleanroom, and the Ada Process Model methods • Formally defined processes for Application Engineering (AE) • Launched a major metrics initiative • Automated staff hour and defect metrics collection	• Nearing completion of formal definition of CM process • Beginning formal definition of Domain Engineering Process • Working on second round of AE specification process • Using automated process modeling and enactment support for SCAI Release 2 and 3
Automated Support	• Commitment to Rational Ada support product-line • Commitment to Universal Network Architecture Services (UNAS), and RICC for Architectural Infrastructure	• Integrated a state-of-the-art open systems SEE: IBM and Sun platforms, Rational and TRW toolsets • Installed advanced process support toolset; encoded and began automated enactment of SCAI Release 2 and 3 processes • Instituted automated tracking capability for problems, action items, etc. • Began use of Rational SoDA for automated document production	• Enhancing the functionality and integration of the process tools • Applying Amadeus to automate collection of SEE usage metrics • Extending process automation across geographical locations

Table 4 - Air Force Demonstration Project - Megaprogramming Progress and Status

1.3.2 Introduction to Experience Areas

This Experience Report includes five major sections which provide the Demonstration Project's experience during the Preparation Phase and the first year of the Performance Phase:

"Section 2.0, Domain Engineering/Application Engineering"

A central theme of STARS megaprogramming is enabling systematic reuse across a product-line of applications. STARS advocates that the product-line organization institute a Two Lifecycle process model, with Domain Engineering determining how to exploit the commonalities across the applications, and Application Engineering developing each application in accordance with the common approach and common assets identified by Domain Engineering. One of the key lessons learned by the SCAI team during the Preparation Phase is that Domain Engineering and Applica-

tion Engineering are identical in many respects - so much so that it proves awkward to discuss them separately. For this experience report, they are treated as a single topic.

Prior to the formation of the partnership with STARS, the SWSC had already recognized the importance of domain-specific reuse and had launched a number of informal Domain Engineering activities. One of the key objectives of the partnership was to transform these informal activities into a systematic process. During the Preparation Phase, the team succeeded in formulating the SCAI DE/AE approach and defining an overall process architecture. This process was applied and refined during the first year of the Performance Phase.

"Section 3.0, Process Support"

The STARS concept of megaprogramming calls for concerted attention to process: in order to enable a product-line with effective domain-specific reuse, a systematic process (including a process-improvement process) is required.

Whereas the prior section discussed the work done to define and use the Domain Engineering/ Application Engineering process, this section deals with the technology used to support the process definition and enactment.

"Section 4.0, SEE Support"

Megaprogramming requires a significant level of SEE support to provide automated assistance to the product-line's process definition and enactment. During the Preparation Phase, Loral worked with the Air Force to procure and install two major SEE increments, basing the acquisitions on early understanding of the SCAI process and the sizes of the teams involved in the various engineering and support disciplines. Also during the Preparation Phase, Loral continued work on its STARS-sponsored Process Support Environment (PSE), which is intended to provide a process-driven basis for SEE integration.

During the first year of the Performance Phase, based on a maturing understanding of the SCAI process, the Air Force planned and ordered the third and final SEE increment - and by the end of October, installation had begun. In addition, the SCAI application engineering team began using the PSE, generating needed usage experience (problems, recommended changes) that motivated a second-generation upgrade to the package - which is planned for use for SCAI Release 3 activities. The second SEE survey, conducted in August 1994, indicated that SEE end users believed that the SEE was adequate to support their work and that it was evolving in the right direction to support SWSC product-line objectives.

"Section 5.0, Metrics"

Metrics are instrumental in providing information to quantify software lifecycle cost, quality, and capability differences resulting from the presence and absence of megaprogramming technologies. The SCAI project will quantitatively and qualitatively measure the effects of megaprogramming on the development effort and the resulting product. During the Preparation Phase, the SCAI team formulated a preliminary metrics process. The foundation for the metrics process is the formally defined set of project goals. The team defined the project goals (refer to Section 1.1.3, Summary of Demonstration Project Objectives) and began detailing the metrics collection and analysis approach.

"Section 6.0, Technology Transition"

The ultimate purpose of the Air Force STARS Demonstration Project is to transfer megaprogramming technologies into use within the DoD community. The SCAI project's mission is to formalize the megaprogramming approach, demonstrate its effectiveness and transition the technology. There are three aspects of transition to be considered:

- Transition of technology into the SCAI project in order to demonstrate its potential and gain experience for further transition.
- Transition of technology out of the SCAI project, in a focused way, to an identified set of technology "customers."

- Providing awareness of the technology to a wider DoD community via presenting technical papers and participation in conferences.

There are over a dozen different technologies that are being transitioned into the SCAI project to provide the needed support for megaprogramming. The first customers for the transition of the technology out of the SCAI project are the Space and Warning Systems Directorate and United States Space Command.

The following sections address in detail the plans, accomplishments, and lessons learned from the SCAI project. The sections are broken into Preparation Phase experience and Performance Phase experience (during the first year). A final version of this report will be published in early 1996 at the completion of the project.

2.0 Domain Engineering/Application Engineering (DE/AE)

2.1 Introduction

The central theme of STARS megaprogramming is enabling large-scale systematic reuse across a product-line of applications. STARS advocates that the product-line organization institute a Two Lifecycle process model, with DE determining how to exploit the commonalities across the applications, and AE developing each application in accordance with the common approach and common assets identified by DE. One of the key lessons learned by the SCAI team is that DE and AE are tightly integrated processes - so much so that it proves awkward to discuss them separately. For this experience report, they are treated as a single topic.

Prior to the formation of the partnership with STARS, the SWSC had already recognized the importance of domain-specific reuse and had been pursuing a number of informal DE activities. One of the key objectives of the partnership was to transform these informal activities into a systematic process.

During the Preparation Phase, the team succeeded in formulating the SCAI DE/AE approach and defining an overall process architecture. The first year of the Performance Phase resulted in the initial release of a megaprogramming application and the refinement of the domain model architecture. This experience report deals with:

(1) Conducting precursor engineering activities:
- Generating a pilot application and
- Generating domain/application models,

(2) Defining the DE/AE process,

(3) Utilizing the process, and

(4) Developing the first release.

The report also provides an outline of the resulting approach itself which embodies much of the team's overall experience.

The general-purpose techniques being used on the SCAI project to define the DE/AE process are discussed in the Process Support section of this report.

2.2 Long Term Objectives

The following is a recap of the long-range project goals that are relevant to DE/AE.

(1) Product Goals
- Develop a space mission capability suitable for operational deployment by October, 1995.
- Develop reusable domain assets.

(2) Approach Goals
- Apply the RICC technology to SCAI development and demonstrate its benefit.
- Institute cooperating DE/AE processes, including those for ongoing process improvements.

(3) Institutionalization Goals
- Support the establishment of a product-line organization and infrastructure, responsible for managing the evolution of multiple application systems.

2.3 Preparation Phase

This subsection discusses the DE/AE experience gained during the Preparation Phase of the SCAI Demonstration Project.

2.3.1 Plans

Preparation Phase Objectives

Based on the long-term DE/AE goals discussed in Section 2.1 above, the Preparation Phase objectives identified for this area were:

(1) Near-Term Product Objectives

- Verify the RICC-based architectural infrastructure in the Space domain.
- Construct preliminary domain/application models.

(2) Near-Term Approach Objectives

- Formulate a DE/AE approach, including a preliminary process architecture.

(3) Near-Term Institutionalization Objectives

- Improve domain expertise.
- Learn the technologies and tools to be used in the DE/AE approach.

Assumptions

The following were some of the key assumptions made by the team at the outset:

(1) Use and improve the existing architectural infrastructure approach.

The 1990 AFSPC drive to develop a strategy to move to an architecture based on Open Systems Environments resulted in the development of a set of architectural components, called the C^2AI. Leveraging work accomplished by TRW on a production contract called CCPDS-R, the SWSC contracted TRW to advance CCPDS-R techniques and demonstrate a subset of the Cheyenne Mountain Missile Warning requirements, on a pilot program called RICC.

Significant commonality exists between C^2 systems, and the RICC architecture infrastructure developed reusable software to address these common requirements. The informal analysis of the C^2 domain resulted in a two layer architecture, with a service layer supported by a set of Ada Program Generators.

The foundation for Domain Specific Reuse for the STARS/SCAI project, consequently, is the RICC architectural infrastructure and program generator technology. See Appendix B for more information on the RICC approach.

A major assumption for the Demonstration Project was to take advantage of the RICC approach - including gathering additional data about its viability in this domain, and improving the capabilities and maintainability of the associated toolset.

(2) Use the modeling work already in progress at the starting point for the domain/application analysis.

Several key modeling activities were already underway prior to the start of the Demonstration Project. The CIM model deals with the general operational requirements for various Cheyenne Mountain command centers, while the MOIA model focuses on the command center operations of the space support systems in particular.

These modeling activities were judged to be essential ingredients in any realization of the SWSC Domain Engineering approach, and the SWSC decided that they would be continued and that the work products would be incorporated as part of the domain modeling strategy.

Refer to Appendix B for a more detailed description of these modeling approaches.

(3) Use the Ada language for all software development.

(4) Continue reliance on the Rational Ada Support Environment.

Given the commitment to Ada, the SWSC had already established a close relationship with Rational and had decided to adopt the Rational Ada programming support environment and associated tools for the SCAI project. At first glance, this appears to be an assumption that is relevant only to the SEE, but it also proved to be influential to certain methodological decisions made during the Preparation Phase - such as the specific object oriented method to be used to develop the space domain model (the Rational Booch method, supported by the Rational/ROSE tool).

(5) Capitalize on certain STARS-sponsored concepts and technologies:

- Domain Analysis Process Model (DAPM),
- Conceptual Framework for Reuse Processes (CFRP),
- The Two Lifecycle paradigm for coordinated DE/AE processes, and
- The Cleanroom Software Engineering Process.

These concepts and technologies are discussed in the following subsections. Also refer to Appendix B for more details.

(6) Iterative nature of Software Engineering Process

The resulting approach also needed to allow for iterative development of systems, with ample opportunities for early buy-in to the design by the customer. The approach must also support product-line development, and be evolutionary in nature.

Planned Activities

Given these assumptions, the activities to be used to accomplish the Preparation Phase objectives were:

(1) Develop the SCAI pilot,

(2) Build initial models in the C^2 and Space C^2 domain:

- Space C^2 operations as currently practiced in Cheyenne Mountain,
- Capability model of the current Space Defense Operations Center (SPADOC) application software, and
- Information model of the current SPADOC application software; and

(3) Develop and document the DE/AE approach.

Table 5 depicts how these activities were intended to address the Preparation Phase objectives.

Preparation Phase Objectives	Planned Activities		
	SCAI Pilot	Initial Requirements	DE/AE Approach
NEAR-TERM PRODUCT OBJECTIVES			
Verify the RICC-based architectural infrastructure in the Space domain	X		
Construct preliminary domain/application models		X	
NEAR-TERM APPROACH OBJECTIVES			
Formulate a DE/AE approach, including a preliminary process architecture			X
NEAR-TERM INSTITUTIONALIZATION OBJECTIVES			
Improve domain expertise	X	X	
Learn the technologies and tools to be used in the DE/AE approach	X	X	X

Table 5. Relationships Between Preparation Phase Activities and Objectives

The DE/AE work can be thought of as two interacting activity groups: activities designed to formulate the approach (DE/AE Approach Definition), and activities designed to develop engineering products (DE/AE). Recognizing the need for usable products at the conclusion of this phase, a conscious decision was made to perform both activity groups in parallel. The early engineering work was intended both to provide experience needed to develop the approach and to produce work products that could launch the production phase work.

It should be noted that in view of the large amount of learning that faced the team, it was not possible to lay out a very detailed schedule. The plan boiled down to fairly loose statements of objectives in each area, with a deadline of 10/1/93 (the nominal start of the Performance Phase).

Some of the most profound objectives were institutionalization objectives: improve domain expertise, and learn the technologies and tools. Each of the planned activities would contribute to these organizational learning objectives.

A key challenge facing the team was the fact that the SCAI project brought together multiple organizations, each with successes in using technologies with which they were familiar. This required that the project members had to understand the variety of SWSC incumbent technologies, the organizations in which those technologies were used, and the STARS technologies, before they could begin the process of integrating the technologies into the SCAI technical approach.

In view of the anticipated difficulty in achieving consensus on the DE/AE approach, the Air Force project management was faced with a dilemma. Working out the approach seemed to require the most talented, most experienced people. Yet if these people were devoted exclusively to the approach definition work, the product objectives could be in jeopardy. On the other hand, in the absence of a well-defined approach,

there was a strong risk that the products developed during the Preparation Phase would not provide a sound basis for the Performance Phase work.

The management approach adopted by the Air Force was to keep its product-oriented activities relatively isolated from each other - and from the approach definition activities as well. Each product-oriented activity would be kept apprised of the maturing approach strategy but would be allowed to evolve its own detailed methodology. Similarly, the approach definition activity would be kept apprised of the results being achieved in the product-oriented activities and would attempt to take advantage of their experience in evolving the overall DE/AE approach. As confidence increased in the maturing DE/AE approach, more interaction between the various groups would be encouraged.

2.3.2 Summary of Accomplishments

The SCAI team successfully accomplished all of the major Preparation Phase DE/AE objectives cited in the prior section. The following paragraphs summarize the accomplishments:

(1) Developed the SCAI pilot

The pilot implements several significant functions present in the existing SPADOC application using the new open-systems based architecture infrastructure. It has served as an excellent communication vehicle to the future users of the SCAI application; it has also provided a concrete demonstration of the viability of the architectural approach to outside observers. Building the pilot has helped transition the RICC- based system construction methodology to additional team members, and it has also helped improve the team's space domain expertise.

The most significant result of the pilot activity is the additional confidence the SWSC now has that the architectural infrastructure is viable for multiple applications within the product-line.

(2) Built initial requirements models:

- The CIM and MOIA models produced during the Preparation Phase capture the space operations to a sufficient level of detail to support the Performance Phase work. These models have served as a good basis for communication with the future users of the SCAI application. (These models are defined in Appendix B, and they are also discussed in more detail later in this subsection).
- Built a capability and information models of the current SPADOC application software, with emphasis on identifying the essential abstractions in the Space domain.

  Two models were developed: an object oriented model and an information model. The models will provide the foundation for the SCAI specification effort during the Performance Phase.

  The initial intent of these models was to capture the existing SPADOC software, to serve as the basis for re-engineering much of the SPADOC functionality for the SCAI. As this modeling activity progressed, however, another key objective emerged: the need to recast the current functionality in a form that would serve the interests of other applications in the domain. This led to the emphasis on identifying essential abstractions in the models, including the decision to map the SPADOC functionality to a new object oriented view.

(3) Developed and documented the DE/AE approach

The team achieved consensus on the overall approach, which was documented in Version 2.0 of the SCAI Demonstration Project Management Plan (SDPMP), dated 10/15/93. The approach is reflected in the SCAI process architecture, documented in $IDEF_0$ form, and attached as an appendix to the SDPMP.

The remainder of this subsection presents the experience gained while accomplishing the above.

2.3.3 Analysis

This material is based on more detailed experience information to be found in Appendix E of this document.

SCAI Pilot Experience

The SCAI pilot was designed to test and demonstrate the feasibility of using the RICC as the architectural infrastructure to support the space mission. Two representative space mission applications were implemented:

(1) Satellite Ground Tracks

(2) Sensor-Satellite Look Angles

The target hardware was the IBM RISC System/6000 series workstation. The intent of the pilot was to implement the applications with the look and feel of SPADOC 4B displays and to have the software produce the same numerical results as SPADOC. Allowances were made to accommodate differences between RISC/6000 and the SPADOC 4B Megatek with respect to keyboards and X Window capabilities, and the pilot was permitted to make minor improvements in display format and functionality.

Kaman Sciences was teamed with TRW. Kaman was responsible for implementing the space application software, and TRW was responsible for implementing the user interface displays, the data base, and the input and output message processing using the TRW developed RICC and associated tools. The application software was coded in Ada.

To determine the functionality to be implemented in the pilot, the team iterated on a set of "threads," or scenarios that were defined in terms of coherent user interface activities. Several reviews were held with current SPADOC users present in order to gain concurrence that the threads defined would be a representative set.

The goal of writing a small but representative subset of space applications software in Ada and of using the RICC architectural infrastructure was achieved. Using satellite element set and sensor location/limit data from SPADOC, the SCAI pilot produced ground tracks, ephemerides, and sensor-satellite look angles for five satellites representing a cross section of the satellite catalog. The results matched SPADOC 4B results within a reasonable degree of accuracy.

Modeling Experience

(1) Operations Modeling

Following the CIM methodology, two workshops were held: a Baseline workshop which established the scope and began the analysis of the 'AS-IS' Space Control Mission, and an Activity Based Costing (ABC) workshop. The Functional Economic Analysis (FEA) workshop which would normally follow the ABC workshop was postponed until after Missile Warning Mission and the mission of the Command Center could be completed to allow the FEA to be conducted on the aggregate processes. The workshops developed, refined and validated the Space Control Mission 'AS-IS' activity model, identified improvement opportunities related to the processes performed in executing the mission, developed a Space Control Mission 'TO-BE' activity model incorporating improvement opportunities. This effort was accomplished by a team of users and SCAI project team members and was viewed as top-down perspective of the operational activities.

The MOIA approach of analyzing command center operations through detailed reviews of concept of operations, checklists, and other command center artifacts was applied to three distinct space related operations centers in CMAS, 1st Command and Control Squadron (1CACS), Space Surveillance Center (SSC) and SPADOC. The effort was recorded using the IDEF notation and was viewed as bottom-up.

The CIM model and the MOIA models were then compared to ensure that the top-down and bottom-up approaches merged. These effort resulted in two major accomplishments:

- The development of an operational model of the space mission which could be used for both the applications development and further understanding of the Space and Warning domain.
- The opportunity to acquaint the user with the STARS Demonstration Project

(2) Space Domain Requirements OO Model

The process used by Kaman Sciences to build a Space Domain Requirements Model was developed incrementally and is a composite of the Booch, Shlaer-Mellor, Concept Maps, and DAPM (refer to Appendix B, Technologies Contributing to SCAI). Using existing SPADOC 4 system documentation, phase one constructed an unabstracted object oriented model of the existing system. Phase two mapped the unabstracted model into two layers: a service layer and an application layer. Each layer was comprised of a pseudo-Booch Logical Model. This two-layer model was consistent with the RICC architecture. It also met the DAPM starting point requirements of postulating an initial architecture for the domain of interest. The third phase, with reuse as an objective, identified new services, when analyzing redundancies in the unabstracted Concept Model, that could be mapped from the application layer into the service layer. The Booch Logical Modeling, used during the second phase, was modified. Ada program design language (PDL) was added to describe the behavior of the objects identified in the model, and to have the models be consistent with Cleanroom practices.

The need to exclusively use the DAPM formal domain analysis technique to model the space domain was deemphasized because the project postulated that from a functional software perspective the operational system (SPADOC 4) software was equivalent to the Space Control Domain. This assumption is reasonable because SPADOC 4 is a replacement for existing SSC systems, and is by definition a superset of the Mobile system. As a result, the focus was to create AE artifacts that were consistent with DE principles and could be easily generalized into DE artifacts. The process design team was to ensure that the AE process was reusable as a DE process with a broadened scope of the analysis.

(3) Space Domain Requirements Information Model.

The information modeling effort began with extracting database design information from the SPADOC 4 system engineering documentation. That information was analyzed and used to develop a third normal relational database schema. A data dictionary was established and populated. The effort was developed using Entity Relationship Diagram (ERD) notation. Due to the vast amount of data to be organized, the initial conceptual data schema has focused on defining all entities, relationships, keys and attributes for release one.

The Space Control Mission workshop also developed a data model using $IDEF_{1x}$ notation. This high-level model was developed based on the information flow depicted in the activity model. This activity proceeded independently of the ERD effort.

The ERD model was compared against both the entities defined in the OO model and the CIM $IDEF_{1x}$ data model to ensure consistency. Although each of the models depicts varying levels of detail and was developed from different perspectives and purposes, the models were found to be reasonably congruent.

This resulted in an ERD-based conceptual schema which could be developed into database tables to allow the development of the application.

Approach Definition Experience

A great deal of experience was gained as a result of the activities in this key grouping. Because of the volume of experience data generated, only a summary is presented here. Appendix E, Preliminary DE/AE Approach, provides additional information for interested readers. The appendix covers the following topics:

(1) Selected DE/AE Approach Topics

(2) Creating a DE/AE Process

(3) Preliminary SCAI DE/AE Process

The following is a summary of the DE/AE Approach Definition experience gained during the Preparation Phase:

(1) Iterative Buildup of Approach

Forging a megaprogramming DE/AE process was perhaps the most difficult technical challenge faced by the SCAI team. This was due to the amount of learning required by each team member, and the difficulty in integrating the best of the strong candidate technologies, methods, and tools being considered. It proved impossible to define the process in one massive intellectual leap. The only way for the team to cope with the problem was to iterate. In retrospect, there was no formal process followed while this work proceeded. Working groups would alternate between periods of intensive interaction and periods of relative inactivity - while other more pressing issues were addressed. Frequently, several approach issues were being studied in parallel by separate groups. Discussions were sometimes contentious. In general, with so much to integrate, a great deal of "groping" occurred.

Despite the seeming confusion, however, some very bright and experienced people managed to make a great deal of headway, and at the conclusion of the Preparation Phase, the team had achieved a high degree of consensus on the overall approach.

Although there was no formal plan or process followed while this iteration was in progress, some patterns did emerge. In general, the implicit process was something like the following:

- Postulate a facet of the approach and attempt to elaborate it;
- Identify the most pressing issues associated with the postulated facet;
- Attempt to resolve the issues;
- If rough agreement can be achieved, attempt to articulate the agreement and document it or present it to a wider audience to gain consensus;
- If a sufficient level of consensus is achieved, attempt to merge the new facet into existing materials used to capture the approach (papers, charts, etc.).

By the end of the Preparation Phase, the approach was far from complete, yet sufficient agreement had been reached about the key aspects of the approach, that it was possible to launch the Performance Phase work.

In summary, the team recognizes that iterating on the approach is a fact of life, and further, that the iteration will continue throughout the life of the product-line (i.e., process improvement is a continuing responsibility).

(2) Coming to Grips with Terminology Problems

One of the key problems impeding convergence on approach was terminology. Terms such as "architecture," "domain," and "model," had different meanings to virtually every person. A frequent scenario was a protracted, sometimes emotional interchange that culminated in a realization that the parties had been talking about two different things (or more!). A glossary would have been very useful, and in fact one gradually emerged. The difficulty in generating a glossary was that some terms continued to defy consensus. A practical glossary appears in Appendix A. The move towards a complete glossary is continuing into the Performance Phase.

(3) Shlaer-Mellor Pilot Model

In an attempt to distinguish exactly what Domain Engineering might be for the SCAI project and to determine exactly what a Domain Requirements Model and a Domain Architecture Model would look like, a four week effort was spent building an experimental Domain

Requirements Model using DAPM and the modeling views advocated by Shlaer-Mellor Object Oriented Analysis. The subdomain of Space Operations was selected for the modeling pilot, since the first release of SCAI software was intended for that domain.

The Shlaer-Mellor methodology advocated a layered model framework, which was also being advocated by the space analysis work being done. This notion of layering ended up being an essential aspect of the current approach (refer to Appendix E).

(4) Resolving Dissonant Aspects of Approach

With the number of technologies, methods and tools being considered, it was inevitable that aspects that were considered "mandatory" appeared to be unalterably dissonant.

As an example, the team had an objective of taking advantage of three key technologies: Ada Process Model, OO Analysis and Design, and Cleanroom Software Engineering (refer to Appendix B, Technologies Contributing to SCAI, for an explanation of these terms). Each had undeniable advantages. Ada Process Model stressed building the system incrementally, with each increment disclosing more of the design, and each increment giving more insight into the true requirements. This seemed to be at odds with the Cleanroom principle of getting the requirements correct from the start, before proceeding to implementation. Cleanroom seemed to dictate functional decomposition, in opposition to the object oriented approach. And so on.

In fact, the team was able to forge an entirely new approach from the three - potentially strengthening each of the ingredient approaches. Although this integrated approach is still being elaborated, and although a lot more experience needs to be gathered, the marriage appears off to a good start. Refer to Appendix E, Preliminary DE/AE Process, for a high-level summary of the current approach, showing how the three technologies are used in concert.

Some of the other aspects of the approach that proved to be hurdles were:

- Mapping the STARS "Two Lifecycle" process model to DE and AE

  Early thinking called for two entirely distinct processes: DE, and AE - each with its own set of artifacts that were somehow tied together. At this point in time, the team believes that the processes are integrated as one process (hence the name of the current Experience Report section), and that many of the modeling artifacts are joined together.

- Requirements Models vs. Architecture Models

  Early thinking called for four distinct models: Domain Requirements Model (DRM), Domain Architecture Model (DAM), Application Requirements Model (ARM), and Application Architecture Model (AAM). The above paragraph indicates that the distinction between the Domain and Application models is blurred, and it now also appears that the distinction between the requirements and architecture models is also blurred.

(5) Capturing the Approach

Once a decision had been made about the approach, the question of how to record it emerged. In fact, much of the approach emerged on slips of paper, on whiteboards, on presentation charts, and (sometimes) in more formal white papers. All too often it occurred that two parties left a working session with a secure feeling that agreement had been reached, only to discover later that there were still differences. Writing the agreement down helped prevent this, but even that wasn't an ironclad solution.

Two main project-wide artifacts have been used effectively to help cement the team's approach. The first was the SDPMP. Version 2.0 of this document, dated 10/15/94, brought together the overall approach in a fairly coherent fashion. The second was the SCAI Process

Architecture, an $IDEF_0$ model of the top three-or-four levels of the SCAI process. This described the activities in the process as well as the primary interfaces among them; it also included a glossary of terms used in the process model.

As the team moves into the Performance Phase, the approach is being further refined into detailed process definitions. This refinement is being done incrementally, with each increment dictated by the timing of the Performance Phase activities and by the maturity of the area of the process. The first process area scheduled for detailed process definition is the specification process - a process that is heavily dependent on the modeling artifacts.

2.3.4 Lessons Learned

The DE/AE lessons are divided into the following subcategories:

(1) DE/AE Process Definition

(2) Creating a Space Domain Requirements Model

Domain Engineering Process Definition

The following lessons were learned while trying to create a DE process that embodied STARS concepts as well as existing SWSC technologies.

Lesson: *Interpret the STARS "Two Lifecycle Model" as a single integrated DE/AE process.*

The SCAI team has found it necessary to re-interpret the STARS "Two Lifecycle Model" to show very close interaction between DE and AE, in order to constantly validate the domain models against real applications in the domain. The models must be developed iteratively to avoid the creation of complete but unvalidated models. In fact, the team has developed a working hypothesis that application level models to a large extent should be views of domain level models.

Lesson: *Domain analysis principles should be applied even to individual applications.*

Domain Analysis (DA) typically looks at multiple systems with the intent of discovering the similarities and the differences. The overlap of DA with Systems Analysis should be in identifying system similarities, which is synonymous with identifying high level abstractions. If the DA process is well-understood it should be applicable to a single system. If DA is applied to a non-perfect, single system, the result should be a simpler, well-structured, single system. Thus, even if an organization is initially unwilling to invest in DA and generalized component development across the whole extent of the domain, they may be willing to apply domain analysis to a single system, and still be able to judge the economic and technical value of the DA process. This does not change the purpose of domain analysis from identifying commonalities across multiple systems.

Lesson: *Domain analysis must include operational analysis of system usage.*

Domain Analysis typically examines similar software systems by examining the requirements, design, and software of these systems. However, to create truly abstract domain requirements (problem-state model), the scenarios via which users interact with the systems must be understood. The entire process that the organization follows developing software systems may also need to be examined in order to create an abstract Domain Requirements Model. For example, in the Space Domain, several systems involve the Orbit Analyst in a complicated dialogue across multiple display screens in order to determine the orbit of a satellite that requires non-routine calculations. Two reasons for creating the dialogue are: 1) The calculations are too expensive for obsolete hardware to handle on a routine basis, 2) Compute-expensive routines would have to be written to duplicate the intellect of Orbit Analysts inspecting on-screen graphics. As hardware becomes faster, the need to involve the Analyst in an external dialogue may disappear. However, without examining the Orbit Analyst's role in the overall organizational process, the reason for the specific Human-Machine Interfaces, which are common across all the Space Systems, would not be clear.

Lesson: *The specific DA process may not be as important as planning and assigning domain experts and systems engineers to search for system commonalities.*

Without a formal process, a set of common services for C^2AI systems was developed by TRW. These services are embodied in the RICC products. Independently, Ruben Prieto-Diaz applied his DA process to the same domain at Contel. Independently, IBM developed its CCS-2000 Domain- Specific architecture. In all three cases, the *same* common services were discovered. The lesson that might be learned is: The specific DA process used to uncover the common services is not as important as the fact that there is a planned and concerted effort to discover system commonalities.

Lesson: *Class models can be used as domain models.*

Booch Class Models are good mechanisms for recording Domain Requirements Models. Superclasses, class categories, and polymorphism intrinsically identify high level abstractions/system commonalities. Sub-classes identify system differences.

Lesson: *Domain analysis should produce the simplest reusable system.*

Domain analysis should not just statically identify reusable artifacts, but should identify reusable artifacts that will result in the simplest possible reusable system in the domain. One method for identifying the simplest possible system is to compare the behavior encapsulated within classes to the interfaces defined between classes, as represented by object scenario diagrams. If identifying a common reusable class results in a scenario of usage that sends a single message into and out of a common reusable class for all systems in the domain, and the reusable class abstracts complicated internal behavior, then the reusable class is loosely coupled. If, however, the scenario of usage for the reusable class across all systems in the domain results in hundreds of messages in and out of the class, then the class is tightly coupled to other classes and is not a good choice as a potential reusable component abstraction.

Lesson: *Domain Requirements Model for C^2AI must be layered.*

Two philosophies of domain-specific reuse existed within the SEI: the D'Appolito school, which believed in basing reuse on a common Domain Architectural Model, and the Cohen school which believed in basing reuse upon both a Domain Requirements Model *and* a Domain Architectural Model. Through practice, we accepted a Cohen Approach modified by Prieto-Diaz. A requirements model shows what problem systems in the domain are supposed to solve. When technological changes mandate a different architecture, systems need not be redeveloped from scratch. Big DoD systems are going through a technology shift right now with networked workstations replacing large mainframe computers. The next such technology transition might be the use of massively parallel computers. Though it is necessary to capture the domain problem in a mode using the Prieto-Diaz process of domain analysis, domain "artifacts" will be uncovered which do *not* relate to the problem state. As an example, references to querying a relational data base are *not* problem state artifacts, but *are* encountered frequently during analysis of C^2AI systems. The Prieto-Diaz approach is to postulate an architecture and provide a layer in which these references can be captured. The postulated architecture for the SCAI is the RICC "Chip" layered architecture. This layer need not be elaborated as part of the formal requirements model, but it can be developed and elaborated as part of the Domain Architectural Model. This service layer to the Domain Architectural Model will need to be elaborated to identify whether sufficient services exist to develop the application layer in the domain architecture.

Lesson: *Domain software components should include validation data.*

A Domain Architectural Model defines the context via which reusable components communicate. The STARS SCAI team feels that reusable specifications, validation, testing, and intended usage context are necessary for reusable software components. This contextual data is contained in Cleanroom Six-Volume Specifications, which are generated by the Cleanroom Process. Cleanroom System (Domain) Specifications will define a formal and implementation free, functional set of requirements. These requirements will define the boundary of the system. They will also provide a way of validating the Domain Requirements Model. If the same stimulus history applied to both the Domain Black Box and the Domain Requirements Model produce the same response, then the Domain Requirements Model has been validated.

Creating a Space Domain Requirements Model

In the following discussion, describing initial experiences in applying Object Oriented Analysis (OOA) modeling principles, some non-standard terminology is used. One of the techniques used is called "Concept Maps," which is an informal brainstorming technique in which the modeler is not restricted to fixed object relationships, eliminating redundant entities, or making clear distinctions between entities and attributes. Object Oriented Design (OOD) refers to a Booch logical analysis process, with two caveats: 1) Object Scenario Diagrams do not show whether message passing is synchronous or asynchronous, 2) Object Scenario Diagrams have Ada PDL associated with them to show high-level behavioral abstraction.

Lesson: *Domain analysis must concentrate on capturing the "apparent" aspects of the domain and characterizing its essence, rather than its atomic details.*

Although time constraints are always a factor in projects, the analysis of the entire space domain in the given time-frame was somewhat overwhelming. The time-factor forced us to deal with the "essence" of the domain and, rather than getting bogged down in the seemingly infinite details within the space mission, we succeeded in capturing the entire domain and its essential behavior and relationships. Consequently, we were able to produce a domain-wide OOD that is merely missing a consistent depth of refinement. The level of depth within the analysis and design products is inconsistent. That is, some entities in the analysis model are detailed well and others are not. In the design model, we could not refine all of the class operations down to the same level, nor could we deal properly with concurrence, persistence, or synchronous/asynchronous messages.

Lesson: *Project terminology needs to be shared and defined in a Project Glossary that is well distributed throughout the project.*

Terminology was a stumbling block. For example, when we initially defined the process and the products we were going to produce, we used the terms "model" and "architecture" and carried these for a long time in the project. However, they later proved to be confusing to other SCAI members and thus counterproductive. Although we changed the terms to OO Analysis Model and OO Design Model late in the game (expensive in labor-hours) to try to overcome the difficulties, even these terms are not without castigation, and a consensus on terminology is still elusive.

Lesson: *The flexibility to change process definition real time is important.*

The definition of a process was one of the absolute necessities for this modeling effort. It defined the ordered steps and activities with their accompanying inputs and products. We made several modifications to the process as we were going through it and that in itself is important. Having the flexibility to change the process for real-time improvement is important. It was essential not to treat the defined IDEF process activities as inflexible due to the amount of interaction that existed between activities and the detailed artifact definitions required.

Lesson: *Both functional and object oriented mind sets are required.*

Due to the functional nature of the SPADOC requirements, and the need to depict the control state aspects of the SCAI threads, it proved necessary to use our functional mindset at times. However the learning curve for OOA/OOD is steep and thinking in terms of objects was not easily caught by those who have been using functional decomposition for many years. We were fortunate to have someone on our team who had several years experience in OOA/OOD. This proved to be an extremely valuable asset.

Lesson: *A single or chief architect who is responsible for assuring the integrity of the overall system architecture is essential.*

Note that OOA refers to creating an unabstracted OO Space Model. OOD refers to creating an abstracted Booch Class Model. When we entered the OOD phase, it became very clear that we needed to have an overall architect who would design the initial class diagrams and review all of the class and object diagram development. Developing the DRM would not have succeeded without a chief architect. A chief architect was essential to maintaining a logical and consistent design. We also discovered that every team member had to have a total domain understanding in the OOD phase. This was a change from the OOA phase, where each analyst could take a particular leg of the analysis model and characterize the individual entities. However, in the OOD phase, the relationships between the classes and objects were much more pervasive and required a broader understanding of the domain.

Lesson: *The design process of stepwise refinement is valuable.*

Although not unique to OOA/OOD analysis, we found that following the design process of stepwise refinement very valuable. We developed the initial class diagrams with their operations and the initial object diagrams and then began the refinement process. As we continued this refinement, we discovered new operations, classes, relationships, etc., and simply updated the model accordingly. This process can simply be continued until you arrive at the code level following very easy and small incremental steps.

Lesson: *State transition analysis of operator sessions was valuable.*

One of the unique things that emerged from this effort was the characterization of the sessions through state transition diagrams. It would be hard to overstate their value to this effort. They provided a solution to capturing what appeared to be very complex display and operation relationships. We used these session display state transition diagrams as one of the basic foundations for the OOD phase. Sessions are now viewed as comprising a new layer to our Domain Requirements Model called the "Event" layer. This layer is very closely related to an operational concept for the SCAI system.

2.4 Performance Phase

One of the key priority decisions made on the Air Force/STARS Demonstration Project was to emphasize AE over DE. The prior section (Preparation Phase) has provided some background for this decision; but the primary factor was that earlier work - before the start of the Demonstration Project - had already convinced the Air Force that the RICC C^2 Architectural Infrastructure (C^2AI) approach was viable for most of the applications in the SWSC domain. This earlier work included piloting of two other SWSC application areas (missile warning and Air breathing threat warning) using the C^2AI.

Since the selected Demonstration Project application - the SCAI - was a mobile space C^2 capability, one of the main Preparation Phase activities was to construct a variety of space domain models. These models paid little overt attention to the non-space domains, but rather focused on thoroughly understanding the space application subfamily - from several points of view: the functions of the current CMC space operations centers, the database model for the SPADOC system (the antecedent application upon which the SCAI requirements are based), and an object-based model that attempted to generalize several existing applications. These models were not formal domain models - which resource constraints prohibited - but a great deal of attention was paid to generality in the hope that they could be evolved to accommodate other SWSC applications at a later point in time.

These models were largely complete by the end of the Preparation Phase and served as the starting point for the AE activities in the Performance Phase.

During the first year of the Performance Phase, proportionally even more emphasis was placed on AE - i.e., building the SCAI application - with DE personnel limited largely to refining the DE/AE process definition and to proactive involvement in AE processes. The intent of their participation in the AE processes was to help the AE team adhere to the abstract space domain models and to follow good architectural principles during the construction of the application.

2.4.1 Plans

During the first year of the Performance Phase, the DE/AE team utilized megaprogramming technologies and tools to implement the DE/AE approach to building a SWSC product while at the same time improving the processes and tools based on experience. These objectives translated into the following goals:

(1) DE/AE Process Goals

- Elaborate and refine the DE/AE process, consistent with the SCAI process architecture.

(2) DE/AE Modeling Goals

- Refine the Space DRM begun during the Preparation Phase.
- Construct ARMs and AAMs consistent with the RICC infrastructure, and the DRM.

(3) DE/AE Architecture Goals

- Write an Architectural Framework Document that extends the concept of Domain Architecture principles for use across the wider domain of Cheyenne Mountain Complex (CMC) C^2 Systems.

  The original plan (from the Preparation Phase) to build a DAM was dropped because the potential for the DAM identifying additional reuse opportunities was not significant enough to justify the cost. An Architectural Framework document was added which extends the concept of architecture to the relationship between architecture and all domain and application artifacts (DRM, ARM, AAM, specification, code, etc.).

- Generate SCAI infrastructure components using RICC off-line code generator tools.

(4) SCAI AE Goals

- Develop the System Specification using the Cleanroom Specification process. The goal was to develop a high level system specification with the system black box specified and the component black boxes specified according to the overall system architecture. The overall system architecture was to capture the initial concepts of layering in terms of Black Boxes for the System Infrastructure Layer, the Mission Application Layer, and the Application Service Layer. The System Specification was to provide generic interfaces for abstract categories of stimuli and responses while referring to appendices that delineated the details of messages, displays, stimuli, and responses for the entire system.
- Develop the Release 1 Specification using the Cleanroom Specification process. The goal was to scope the contents of Release 1 according to the published project schedule, develop specific software specifications for Release 1 requirements, and break Release 1 into software increments using the Cleanroom Specification process.
- Develop the Detailed Design for Release 1. The goal was for TRW to develop the detailed design for the displays, messages, database, and user interface; and for KSC to develop a detailed design for the mission code, using the object oriented model. The detailed design was developed using the hybrid process of Cleanroom Engineering integrated with the TRW Ada Process Model and integrated with object oriented analysis and design. This process is referred to as the SCAI Incremental Software Development Process.
- Develop the screens, messages, displays, mission components, software architecture skeleton (SAS), and database for Release 1. The goal was to develop the software increments according to the Cleanroom Specifications as approved by the Specification Team. The Development Team was directed to use the SCAI Incremental Software Development Process referred to above to the highest extent possible without redoing work accomplished during the Preparation Phase.
- Integrate Release 1 increments. The goal was to integrate each software increment in preparation for the Certification Team to test it. The Certification Team was to test each increment as the next one was being developed.
- Certify Release 1 increments. For Release 1, management decided that Cleanroom usage-based, statistical testing would not be incorporated into Release 1 testing. The team was directed to use thread testing similar to what was used to test the pilot. Management made this decision because they felt that the team did not have the required resources (staff hours and time) to successfully incorporate Cleanroom technologies into both development and testing.
- Develop the Release 2 Specification using the Cleanroom process. The goal was to develop the Release 2 Specification with the enacted Specification Process developed by the Process Team. The team was directed to begin using ProjectCatalyst as the automated tool for supporting the enacted process.

- Develop the screens, messages, and displays for Release 2. The goal was to develop these components with the enacted Development Process developed by the Process Team. The team was to integrate the process tools into the development activities.

2.4.2 Summary of Accomplishments

The following is a list of accomplishments from the first year of the Performance Phase:

(1) DE/AE Process

- Refined process architecture ($IDEF_0$ depiction) for DE and AE subprocesses.
- Completed formal process definition packages for the following AE subprocesses: "Prepare Specification", "Incremental Development" and "Certification" (using process definition techniques discussed in Section 3.0, "Process Support").

(2) DE/AE Modeling

- Developed the Space Domain Model (draft) scoped to cover the Space Control Domain.
- Developed the SCAI ARM and AAM.

(3) DE/AE Architecture

- Developed internal copy of Architecture Framework document.

(4) SCAI AE

- Developed the SCAI Cleanroom System Specification.
- Created SCAI Release 1 consisting of 332K lines of code (see Section 1 for details). Release 1 accomplishments include:
  - Developed Cleanroom Release 1 Specification.
  - Developed detailed design.
  - Developed Screens, Messages, Application Software Architecture Skeleton (SAS), Mission Components, and Database.
  - Integrated all seven software increments.
  - Developed Certification Plan and Test Cases.
  - Integrated all seven software increments.
  - Demonstrated Release 1 SCAI system to USSPACECOM and STARS.
- Release 2 accomplishments:
  - Developed Cleanroom Release 2 Specification.
  - Started development of Screens, Messages, SAS, and Database.
  - Started detailed design.
  - Started code development.

2.4.3 Analysis

This section discusses the experience of the first year of the Performance Phase, organized by the same four categories used above. Please refer to Appendix I, SWSC Domain Engineering Experience, for background on modeling needed for understanding the narrative presented in this section.

2.4.3.1 DE/AE Process

Considerable attention was given to the AE aspects of the AE/DE process during this period, but little concerted attention was given to the DE aspects. This was a conscious decision, since the project had chosen at the outset to focus on AE and rely on the previously conducted informal DE work which yielded the

SCAI architectural approach. The hope was that additional DE resources could be allocated later in the second year of the Performance Phase.

Several AE subprocesses were specified and modeled using techniques described in Section 3.0, Process Support. These subprocesses are discussed in some detail in later subsections, starting with Section 2.4.3.4 on page 32.

The SCAI view of the STARS Two Lifecycle Model matured during the Preparation Phase into a set of very tightly coupled iterative Application and Domain Engineering processes, with highly restricted roles for each lifecycle. The role of DE is to extend the scope of the DE domain analysis eventually to the entire set of SWSC Command and Control Systems. The role of AE is to build both system-specific and domain-general code identified by AE. The reason that domain-general code generation is assigned to AE, is that it must be tested across all scenarios of usage (understood by AE), against the application-specific requirements that are maintained by AE.

A series of planned changes to the AE/DE process were identified, but never formally encoded into PDIOT[1] because insufficient staff was available. Following are some of the changes that are planned, given that staff becomes available:

(1) The System Engineering Process must be elaborated to identify the way the Service Layer of the DRM is translated into a set of Application Specific Services.

(2) The Application Architectural Modeling process must be elaborated to explain the trade studies that are performed to determine how Missions from the Service layer are distributed across the UNAS tasking structure.

(3) The process for modeling in the Session DRM layer and its relationship to the Application DRM Layer, must be elaborated.

(4) The Domain Management Activities related to Configuration Management of the Models and Code Artifacts must be elaborated.

(5) The Domain Engineering Activity for extending the Domain Model to include new Classes must be elaborated.

2.4.3.2 DE/AE Modeling

During the Performance Phase, funding has not been available to extend the scope of Domain Analysis beyond the SWSC Space Domain. Also, the Protect Mission subdomain of the Space Domain was kept at a highly abstract level.

DE continuously uncovers new abstractions within the current scope of the domain, resulting in more common code. DE records new abstractions (classes) in the DRM. SCAI has decided to rename the DRM as the Domain Logical Model (DLM) because it is actually a high level logical design for the set of SCAI systems. No other artifacts are maintained by DE. Physically, the DRM and ARM will eventually have to be maintained as separate files, so that instantaneous images of the DRM/ARM can be captured that correspond to the delivered versions of the SCAI Application.

The ARM (to be renamed as Application Logical Model or ALM) contains the same set of classes as the DLM. The reason that application-specific classes are maintained in the DLM is that as the scope of the Domain Analysis is extended, these application-specific classes may be generalized so that they apply to more than one system. AE adds detail to existing application-specific classes. The details of application-specific classes are complete whenever the code increment that includes those classes has been delivered and tested. The AE activity has continuously elaborated new classes associated with unelaborated Missions in the DLM.

If resources become available, DE plans to add a User Interface Layer to the DLM, that will capture User Activities associated with Space Missions. Currently, User Displays are captured in SCAI Missions, but not the activities performed by the User. Currently, Mission Specific Classes that relate to building individual

1. Process Definition Information Organizer Templates - explained in Section 3.0, Process Support

displays, exist in the Mission Layer. These Mission classes could be abstracted further when the User information is also subject to DA.

The Domain Architecture Model (DAM) was dropped from SCAI plans because its payoff, in terms of reuse provided, versus the cost of developing and configuration managing the artifact, was deemed low. Also, the Booch OO Methodology, adapted late in the Preparation Phase, does not contain the mapping templates that would have made the construction of a DAM a natural activity in the method advocated by the DE team.

Information Modeling and Database Design

Early in the SCAI evolution, one of the SCAI contractors (PRC) was tasked with providing an 'AS IS' conceptual data model of the Space Control environment as defined by SPADOC 4C, Version 2. The conceptual data model identifies, describes, and provides data attribution for the entities and relationships that provide the informational view of the space control mission. The objective of the data modeling effort is to create a normalized database management system (DBMS) independent, conceptual schema, derived from space control requirement specifications. The basis for the space control conceptual schema is the persistent data, defined as being maintained in the SPADOC's data management structure. The resultant space control conceptual data model in turn provides the conceptual basis for the SCAI database design for follow-on implementation.

To develop the conceptual data model, the modelers reverse-engineered the current SPADOC file and data field descriptions into an ERD-based model using Cadre Teamwork/IM. The supporting data dictionary for the data elements, entities, and relationships exposed during the analysis is based on guidance from DoD Directive 8320.1, Department of Defense Data Administration, and DoD 8320.1-M-1, Data Element Standardization Procedures. Full compliance with the DoD Data Model and the Defense Data Repository System (DDRS) was not attempted, since neither was available to the data modelers, and since full compliance was beyond the allocated SCAI effort. The initial release of the conceptual data model, commonly called the "core model", addresses the "core" of space control - namely satellites, observations, element sets, ground sites, sensors, systems, etc. - and the relationships among them. The initial release of the data model was a superset of what was necessary to support catalog maintenance - the prime capability provided by SCAI Release 1.

Release 2 of the space control conceptual data model is under development and will more than meet the requirements of SCAI Release 2. Driving toward a model of the space control "big picture" allows for understanding the whole problem domain, flexibility in accommodating release implementations, and better integration and interoperability with other functional areas in the future.

To parley the conceptual model into a database design, a de facto database working group formed to review the information needs of the SCAI and determine which portions of the data model would be necessary to provide the basis for the SCAI database implementation. After review with developers and the Government, those entities and relationships needed for Release 1 were identified. The resulting SCAI conceptual schema provided the basis for the structured query language necessary for creation of the database schema - which was then modified to meet the specific needs of the SCAI and maintain compatibility with SPADOC.

2.4.3.3 DE/AE Architecture

The SCAI Architecture currently has three layers (Mission, Application, and Service), and three views (Functional, Logical, Physical), creating a matrix with nine "entries". The SCAI Architecture Framework defines each of these "entries" and describes the process which is used to elaborate each of these entries. A SCAI lesson learned is that there is an intimate relationship between process and architecture. SCAI has observed that as the architecture has evolved, so has the process for supporting that architecture. See Appendix I describing the Architecture and Architecture Framework.

Tersely, the Functional view of architecture is elaborated via Specifications and Box Structures ala Cleanroom Software Engineering Methodology, and is Application Specific. Black Box Subfunctions in the Functional View of the Architecture relate to Missions in the DLM. The Logical View of the Architecture is maintained in the DLM, and has a scope of the Space Domain. The Physical View of the Architecture exists

at the Application Level. The Physical View of the Architecture (the Application Architecture Model) is comprised of a UNAS network model, as well as a table which shows how logical Missions are mapped to UNAS Tasks and to processors in a network.

Architecture Working Group (AWG)

The AWG was formed shortly after the beginning of the Performance Phase. Its objective was to implement the integrated DE/AE process described in Appendix E. The initial work of the AWG emphasized support for AE. With membership from the DE, AE, Rational, and Loral teams, the AWG successfully worked issues that were critical to the practical success of the AE team. These issues were also integral to the building of the reusable layered architecture that the DE team is working to define. The issues dealt mainly with how the developers were to implement the three layers described in the systems specification including what components reside in which layer, where is the boundary between layers, what is the specification for the interface between these boundaries. For the DE team, the goal was to promote to the domain engineering models, the architecture developed during the Release 1 AWG activities.

The AWG provided direct support to AE by providing practical guidance on the development of the SCAI SAS. The AWG gathered information on SPADOC system performance, user requirements for concurrent processing, and SPADOC systems architecture. This information was used to decide which applications would reside on the server and which would reside on the individual workstations. A prototype SAS was developed and Rational and TRW performed preliminary performance testing with UNAS. Other issues addressed by the team include dynamic and static task allocation, how the architecture would handle multiple sessions, how the other layers would interact with the RICC tools, how the OO components would interact with the functional components, how classes would be mapped to the SAS, and how to handle time-queued events. All of these practical issues were critical to the success of Release 1, yet they are also important to the long-term development of a reusable architecture for the SWSC.

2.4.3.4 SCAI AE: System Specification

To begin the Performance Phase, the Applications Engineering Team was divided into teams for system specification, release specification, development, and certification. One government team lead was assigned for each area. The Applications Engineering Branch Chief was the lead for the specification and development teams, and the Certification Team had a government team lead from the testing division. The team leads were changed during the course of the period covered in this report. This created some discontinuity in the management and direction the team members received. It is very hard to determine any explicit negative impact that these changes in personnel had on the productivity or schedule of the teams, however, it is the impression that there was some.

The Specification Team started the development of the SCAI System Specification in October, 1993. The SCAI management team decided that the Specification Team would develop both a high level system specification, and three lower level release (or detailed) specifications. The final specification package would contain the overall System Specification and the three release specifications would be combined into one detailed specification. This approach was chosen in order to support the requirement to develop the system in such a way that it would support three distinct demonstrations and because a layered architecture with communications between the layers was desired. This layered architecture became part of the System Specification. This allowed the team to define abstractions of the systems stimuli and responses and to define rules for how to handle each abstraction. This reduced the amount of redundancy in the specifications because the team did not have to address how to handle each stimuli and response individually at the system level. Instead, the team referred to the abstracted description for each type of stimuli and response until the development of each release required more detail.

For the System Specification and for the Release 1 Specification, the Specification Team lead assigned one individual to be responsible for each volume of the "Cleanroom Software Engineering Six Volume Specification". That person, the "book boss", was responsible for the majority of the work in each volume, and for pulling the finished product together. The team held status meetings twice each week to talk about issues, to review completed work, and to approve changes.

Requirements Analysis: During the system specification process, the team encountered difficulties with the requirements analysis because no formal concept of operations existed for the mobile space control mission. As a result, the team decided to base the SCAI requirements on the existing system (SPADOC).

During the Preparation Phase, a Space Control $IDEF_0$ Model was developed to model the space mission processes, discover improvement opportunities, and to understand the space mission domain. In addition, this model was supposed to facilitate requirements definition because it identifies activities, inputs and outputs to the system and how those interact with the process. Unfortunately, the Specification Team did not participate in this modeling effort and were not familiar with the IDEF technique and the advantages to using the Space Control $IDEF_0$ Model. As a result, the Space Control $IDEF_0$ Model was not used to the extent that it could have been for requirements definition.

Another complication in this issue is that the Space Control $IDEF_0$ Model was created for the space control mission in Cheyenne Mountain Air Force Station (CMAS), not the CMAH. As a result, it does not accurately represent operations in a mobile environment, which is the target mission for the SCAI project.

Cleanroom Specification Modifications

The team felt that the Cleanroom Specification Process did not formally address some areas that needed to be addressed in a system specification. The team added sections for data base concepts, security requirements and performance constraints. Requirements traceability was an issue that the team felt was not addressed well by Cleanroom. The team decided to add information about which release each requirement would be implemented in, how each requirement would be tested, which increment the requirement would be implemented in, etc.

The team baselined the System Specification in January, 1994.

2.4.3.5 SCAI AE: Release 1 Specification

During the Release 1 Specification writing, the Specification Team worked with the Process Team to define a specification process that would be enacted in the Release 2 Specification time frame.

Again, book bosses were assigned to each of the volumes in the Release 1 Specification. The team held status meetings twice per week. As part of writing the specifications, the team was charged with determining how much functionality to implement in Release 1, based on the amount of time and resources that were available. The team scoped the work based on the lines of code that were implemented for similar functionality on the SPADOC system. This comparison is not extremely useful, however, because of the fundamental differences between the old SPADOC code and what was to become the new SCAI code. The number of displays and messages was also used as a scoping metric. The high level of personal experience in the mission area on the team was also used to make decisions on release content. However, the team lead felt that the scoping process used for Release 1 was not based on a very cohesive and repeatable process. As a result, the Specification Team overscoped Release 1, and the scope of the build was later reduced during development.

Process Improvement Opportunities

After the Release 1 Specification was baselined, the team met to discuss improvements to the process for Release 2. These improvement opportunities were discussed with the Process Team and implemented as part of the SCAI Specification Process. The following describes the process improvement opportunities that were implemented:

(1) Organizing to develop Cleanroom specification volumes

A Cleanroom specification consists of six volumes:

- Volume I: The Mission
- Volume II: User's Reference Manual
- Volume III: Black Box Function
- Volume IV: Black Box Validation Argument
- Volume V: Usage Profile

- Volume VI: Construction Plan

The team felt that while having book or volume bosses was appropriate to ensure that each volume was pulled together properly, therefore, the Cleanroom volumes were each written mainly by one person. This caused workload to be unevenly distributed. For instance, the Volume 1 book boss must complete Volume 1 before Volumes 4 and 6. The Volume 2 book boss must complete Volume 2 before Volumes 3 and 5. In addition, each book boss became an expert on one specific volume, but had little expertise in the skills required to write the other volumes. If one member of the team was absent, the work on that volume effectively stopped until that person returned. The team felt that the process would be more efficient if each member of the team wrote portions of each volume and this approach is to be used on future specifications.

(2) Specification volumes

Another improvement opportunity identified and implemented was the format of the specifications. The team had followed the basic outline from the Cleanroom Guidebooks for the Cleanroom documentation. This outline facilitates a rigorous process based on six volumes of documentation. The team felt that the process was based too rigidly on the format of the specifications, as opposed to the objectives of the analysis required. For example, transaction analysis is accomplished for Volume 2. Volume 2 discusses the system stimuli and responses from a user's perspective. Then Volume 3 discusses a black box subfunction for each stimulus discussed in Volume 2. The team felt that having the transaction analysis and black box subfunction in separate documents was difficult to follow. In addition, to ensure that both volumes were consistent, rigorous checking was performed between Volumes 2 and 3. The team decided to combine Volumes 2 and 3, with the black box subfunction following immediately after the transaction analysis for a certain stimuli. This eliminated the rigorous checking and made the specifications easier to follow. In addition, the user transaction analysis included the development of state transition diagrams that show the transitions from a series of input and output displays required when executing each interactive operational session. These user state transitions are also required to write Volume 5. The team decided to put display state transition diagrams in Volume 2, rather than Volume 5, so that the person reading the specifications would fully understand the transaction analysis from the user's point of reference.

Another area of improvement was apparent in Volumes 1 and 4. All of the requirements were listed in both Volumes 1 and 4. When the team made a change to a requirement, dual maintenance for Volumes 1 and 4 was required. The team decided that for Release 2, Volumes 1 and 4 would be combined, eliminating dual maintenance.

The team made the decision to combine the Release 1, 2, and 3 specifications into one Detailed Specification at the end of Release 3 development. The team decided that it would be easier to maintain the specifications in the long term if only six documents existed as opposed to 18. Also, after Release 3 development, whether or not a function is implemented in a certain release is really not relevant information for long-term maintenance. However, the team decided that it would be beneficial for long term maintenance to keep a separate System Specification as an overview of the system and how the system handles different types of stimuli and responses.

2.4.3.6 SCAI AE: Release 1 Detailed Design and Development

In February, 1994, members of the development team and the process team held a workshop with the objective of defining the details of the integrated development process. As described earlier in this document, a high level process integrating three different methodologies - Cleanroom, OOA/D, and the Ada Process Model, was developed during the Preparation Phase. However, the development team felt that it was not to a low enough level to manually enact for Release 1. During the process workshop the participants defined the mechanical details integrating these processes that would allow the developers to manually enact the process (see Appendix B, STARS Technologies Contributing to the SCAI, for details). The

process team used this information to develop the process architecture and templates for the SCAI Incremental Software Development process in preparation for the automated process enactment support during Release 2 development.

Incremental Software Development

Release 1 incremental software development did not proceed as planned. One contractor did not receive a contract, and did not start work on Release 1 until six months after the projected start date. This meant that incremental testing and certification of the software could not be achieved on-schedule.

Scoping

In the middle of the Release 1 development, the team came to the conclusion that they could not develop the planned Release 1 functionality with the resources and time allocated to Release 1. This identified the need for a more accurate scoping process so that work can be scoped correctly *before* the development team starts detailed design.

Stepwise Refinement and Walkthroughs

The details of the method of Cleanroom box structured decomposition and walkthroughs, as tailored for the SCAI project, were not well understood during Release 1. Questions which are now answered include: What documentation is required at a walkthrough, how frequently should it be held, how many black box to state box to clear box iterations occur between walkthroughs, who attends, who signs off that the refinement is correct, etc?

Unit Testing

Though the Cleanroom Software Engineering process does not include a unit test phase, this cultural shift was too dramatic for TRW and KSC development teams. Though the defined incremental development process did NOT include unit testing, unit testing was, in fact, done in certain isolated cases during Release 1.

The development team's philosophy for Release 1 unit testing was to perform unit testing in a controlled environment so as not to allow the developer to make untreated changes during unit testing. The concept was to allow compilation only after the code had been written and approved by a team walkthrough (except in special cases where the developer was testing a feature of Ada or a complex algorithm). At this point, if an error occurred during unit testing, the error would be logged, and it would be traced back to the original cause (including problems with environment variables, etc.). The developer would go back through the design PDL, the class diagrams, the object scenario diagrams, and back to the specifications if necessary to find the origination of the error. Using this logging process, the team will be able to learn from each error and will have the knowledge to prevent the error in the future.The team has not yet decided whether to do unit testing during Release 2.

Black Box/State Box/Clear Box PDL as Code Comments

Should the development team incorporate the black box/state box/clear box PDL refinements into the actual code as comments?

The team agrees that the information would be valuable for future maintenance of the code. However, the team did not come to an agreement as to how much of the information should be included before it becomes too cumbersome to really be valuable. Initially, except for preamble and comments in the code, most of this information was not kept integrated with the code. This will be reviewed as Release 2 begins development.

User Involvement (External Organizations to the SWSC)

The development team held one user review during Release 1 Specification writing to discuss the planned user interface with the users. Another user review was held at the finish of Release 1 Specification. At this review, the team briefed the users on the contents of Release 1 and provided them an on-line review of some of the proposed screens. The users provided comments on the technical feasibility of the release.

Configuration Management

Currently, the SCAI process team is working to define the configuration management process. As a result, the specification and development teams developed interim processes for the configuration management of documents and developed software components (the system management team provides configuration management for COTS tools). The interim documented configuration management process was fairly straightforward, although the mechanical details were accomplished manually because no tool had been chosen as the overall configuration management tool for the project. The software component configuration management process was more difficult to define because the development team has multiple contractors - some of which are located off-site. An interim process was developed to protect the integrity of the software components on the integration system, but this process is difficult to enact because of the coordination required between contractors. This is an area of concern and it is being worked for Release 2.

2.4.3.7 SCAI AE: Release 1 Certification

At the beginning of the Performance Phase the SCAI management team decided that Cleanroom certification concepts would not be used for Release 1 certification. The team felt that there were not enough resources to successfully learn and implement these concepts in the Release 1 time-frame. The team decided to implement these concepts for Release 2.

The certification for Release 1 will be accomplished by thread testing similar to the thread testing done for the pilot during the Preparation Phase. The Certification Team developed a System Certification Plan based loosely on 2167A concepts and Cleanroom certification documentation. Then the Certification Team developed test cases and descriptions for each session in the Release 1 Specification. The certification schedule was based on certifying each increment as it became available. The team developed test cases for Increment 1, then Increment 2, etc. However, for Release 1 the Certification Team will certify all increments at the end of Release 1. The Certification Team is developing a detailed process for recording discrepancies found during Release 1 certification.

2.4.4 Lessons Learned

Performance Phase lessons are grouped into three categories:

(1) Domain/Application Engineering Performance,

(2) Applying and improving the DE/AE Process, and

(3) Tool support.

2.4.4.1 Domain/Application Engineering Performance

Lesson: *Megaprogramming projects still need to focus on basic engineering and project management disciplines.*

Although megaprogramming brings to the table multiple new concepts to improve the production of software systems, it does not replace some of the basic engineering and project management disciplines. Regardless of technologies, and particularly when inserting new technologies, the training process needs to be identified early on and be carried out on time. People are the ones that make any project work. The technologies and processes being used must reflect the needs of the people. Responsibilities for system development and integration need to be assigned early and documented in a central location to avoid confusion about roles and responsibilities.

Lesson: *Domain analysis needs to be supported by a product line organization.*

The SCAI project conducted domain analysis on the Space Control Domain which included the SPADOC 4C data base. A number of redundancies were uncovered in this data base which had built up over the life of the SPADOC system. Due to the stovepipe organization structure and resulting differences in priorities, it has been difficult to cooperate in this area (which could benefit both SCAI and SPADOC). We believe that structuring based on the product-line approach will enhance the organization's ability to take advantage of

Domain Analysis, increase the focus on commonalities between systems, avoid reporting boundaries and facilitate information sharing between current systems and new ones.

Lesson: *Domain analysis needs to be done incrementally and in concert with ongoing application engineering.*

The cost and time to perform the DA of the Space Control Domain was limited by available resources, and the conceptual data modeling analysis of the space control environment is not complete. However, classes in the application layer of the Space Domain as well as all system threads needed to elaborate SCAI missions, were identified. The definition of classes to support the Protect missions which are not part of the SCAI application have only been developed to a high level of abstraction. Also, some non-reusable application-specific classes which relate to single SCAI missions have not been elaborated. All modeling needed to identify SCAI Ada Packages was completed, prior to any incremental coding effort. The ability to continue to build common software has not been halted, because the DA process allows for incrementally building models. As new resources are acquired, the DA process can proceed, artifacts can be further generalized, and systems in the domain will become more abstract, with less redundant code. The impact of the lack of resources to develop the DRM is that when the DRM is extended beyond the Space Domain, existing classes may have to be generalized, causing potential redelivery of related newly generalized Ada Components.

Lesson: *Define architecture before selecting domain analysis method.*

Booch Domain Analysis was used to identify common classes in the Space Domain Model Application Layer when the space architecture was comprised of two layers: the application layer and the service layer. Later, a functionally decomposed event layer was added to the architecture which captured dynamic behavior associated with space missions. Booch analysis could not identify common abstractions in this functionally decomposed event layer. This is an example of why the domain analysis process is intimately tied to the way the domain architecture is decomposed. It is premature to finalize the domain analysis process before the domain architecture is well understood. In fact, both domain architecture and the domain analysis method should be incrementally developed.

Lesson: *Pattern recognition theory worked for the SCAI project.*

It was presumed that reuse would occur in the Application Layer of the DRM as multiple missions in the Event Layer used the same Application Layer Classes. In fact, as missions were developed, not only was it found that as each mission was elaborated, the same common services were required to support the mission, but also that the services were applied in the same order. Thus, a whole set of reusable services were created, and were encapsulated in Class Categories called Event Control and Event Common. This type of discovery of reuse opportunities (through order of class invocation) is an example of a new theory of architectural reuse referred to in the literature as "Pattern Recognition."

Lesson: *Critical paths of the physical architecture must be incrementally tested.*

The Ada Process Model, incorporated into the DE/AE process, demands that a system, including its architecture, be iteratively built and tested. For example, the original architecture did not distribute the human-machine interface module to each workstation creating too much message traffic on the network. This overload condition was identified because UNAS allowed for architectural testing before any significant application code had been written. This enabled the architect to correct the architecture without impact to the schedule. This is a specific example of the general need to incrementally develop and test the physical architecture.

Lesson: *Missions provide better abstractions than system threads.*

Object Scenario Diagrams (OSDs), created from Booch OO Analysis, identify the order in which messages are passed between objects by testing *single* critical paths through the system, thus validating that class definitions support system requirements. For the SCAI project, OSDs define *all* paths connecting *all* objects for a single space mission (missions are major activities typically defined in an operational concept document such as launch a satellite). Each SCAI OSD might identify as many as a hundred paths through the system. SCAI OSD's provide a useful vehicle for understanding the functional behavior associated with each mission, and reduce the complexity of the Mission layer of the DRM from thousands of OSDs to fifty or so Missions.

Lesson: *Ada Code Generators created significant amounts of reusable code.*

Ada Code Generation, such as that provided by the RICC toolset, has greatly decreased code development time. RICC infrastructure provided reuse across the entire space domain including user interface generation, message parsing/assembly, network control, and data base support. This was used to help elaborate user interfaces rapidly and gain customer acceptance. RICC provided 52% of the Release 1 code (19% reused, 33% generated).

Lesson: *Reuse enables early architecture validation*

The SCAI develops systems based on proposed architectures that are iteratively developed and demonstrated. A proposed architecture is executed using the UNAS/SALE tool. The Architecture Testing starts, in general, before any code has been developed, by inserting "burn" statements which identify suggested processor loads and input/output delays associated with future software components. If most of the components needed to elaborate the architecture are reused and already identified as part of the domain architecture, then demonstrations of the architecture of a developed system are more accurate because "burn" statements can be replaced by real reusable software.

Lesson: *Megaprogrammers should be multidimensional technologists.*

Megaprogrammers not only need skills in development, but also in architecture and process definition and improvement. They must help define the process they use to accomplish their work or the result will be an inaccurate, unused process definition. It is imperative for them to enthusiastically accept the process, and help identify inefficiencies and improvements. Likewise, the software developer needs to understand the domain architecture, and models which elaborate that architecture, so they will recognize the reuse opportunities at their disposal.

Lesson: *Geographically dispersed development teams can retard technology transition.*

The individuals on a megaprogramming development team must help define and subsequently use an innovative process that has never before been used in their organization. If the individuals on the team have conflicting loyalties to parts of the process, it will be difficult to ensure that the process is followed uniformly across the entire team. Particularly, if part of the team is housed in a remote location, visibility into their activities will be minimal, and it may be difficult to determine to what extent the team actually follows the process.

Lesson: *Adherence to megaprogramming principles can help manage geographically dispersed development teams.*

As a counterpoint to the prior lesson, it seems clear that a well-defined process and a sound domain-specific software architecture can greatly simplify the management of dispersed teams. First, if the process is sound, repeatable, and well-defined, it will be much easier to assure that all teams - local and remote - are conducting their activities correctly. Second, a layered architecture and good encapsulation decisions will result in clean lines of separation in the software effort - providing an excellent basis for allocating responsibilities to remote teams.

2.4.4.2 Defining and Improving the DE/AE Process

Lesson: *Do not underestimate the break-in time for significantly new processes.*

A new process was developed to support the AE efforts. The process, a combination of Object Oriented, Ada Process Model and Cleanroom technologies, synergistically incorporates the strengths of each method. Ada Process Model is tailored to Command and Control Systems, develops system architectures, and uses demonstrations to identify architectural errors early in the development process. Object Oriented Analysis is good for understanding domains and for identifying reuse opportunities. Cleanroom is good for developing systems that are accurate to statistically determined levels of accuracy. However, the integration of the three techniques was difficult for *all* groups to master because each development group had to learn at least two new methods. It takes only one radically new method to induce "culture shock." The SCAI team did not allow sufficient time for the cultural learning to take place and for the development team to internalize their understanding of the new methods, and become proficient in the use of the new methods. This "cultural learning period" extended the time that it took to develop Release 1.

Lesson: *$IDEF_0$ is deficient for process definition.*

Formal process definition techniques are important to documenting the process. Although $IDEF_0$ organizes activities in a formal representation, it does not address sequencing of steps and dependencies between process activities. The waterfall like depiction (left to right, top to bottom) tends to leave a default implication (although often incorrect) of sequencing. This lack of a high level graphical representation of sequencing was a serious problem. The Process Definition Information Organizer Templates (PDIOT) partially resolved this problem. To understand the timing between activities the whole DE/AE process must be understood at the lowest level. With several hundred pages of process definition to analyze, this puts an unfair burden on managers attempting to understand the process and make resource allocations. Some personnel developed State Transition Diagrams, associated with all the sub-activities at the lowest level, for day to day use. One specific example of the problem was the "Elaborate Infrastructure Layer" (EIL) process which had been identified, at a very low level, as part of "Create the Application Architecture Model" high level process. Using IDEF representation, the project's Software Engineers had agreed to its location. During performance of the process, it was discovered that the EIL subprocess location was not indicative of its importance. Infrastructure dependencies needed to be better defined in the Specification Process. The first time through the process brought up discrepancies which needed to be rectified before the next process execution of Release 2.

Lesson: *Decision making organizations must be represented in process definition.*

The Architecture Working Group (AWG) was created after the DE/AE process was defined. The group was created to transition Domain Architectural concepts into application design to improve the application reusability. Experiences showed that some decisions made were never acted upon or instituted because of the informal nature of the group charter. For example, the AWG recommended an approach to organizing the file structure within the SEE that mapped to the architecture. This recommendation was never acted upon with the result being that the system ran out of disk space during Release 1 integration work. The AWG should have been formed early on in the project, lead by the Chief Architect, and provided a specific charter and role within the overall DE/AE process.

Lesson: *Process improvement pays dividends during the SCAI project.*

Process improvement did occur as a result of discoveries made during Release 1 performance. Process definition gives visibility and allows for formal review of activities and products such that redundancies can be identified and eliminated. Redundant artifacts/activities were removed and information was added to activity steps to produce a more tightly integrated process. Major modifications were made to the six volumes' Cleanroom Specification process. The improvement, used in the Release 2 specification process, produced a better, more streamlined release specification.

Lesson: *Product-line organizations should be aligned along architectural boundaries.*

Contractors and AF personnel have been working in their own environments in such a way that exchange of information has not been easily accomplished. This is aggravated by the large number of different contractors participating on the project. Each layer of the Space Domain Architecture requires a different set of skills and knowledge. For example, elaborating the infrastructure layer requires skill in network architecture while elaborating the event layer requires understanding of space operations. In order to maximize the reuse potential of these layers the people must have these skills. Project personnel should be organized into teams based on the architecture and be co-located to facilitate information exchange (which would also avoid duplication of the SEE facilities). Such organizations would ensure that reusable artifacts are created as a group and not in a vacuum. Organization structures that parallel megaprogramming architectures enhance the communications needed to maximize reuse opportunities and deliver on the promised productivity increases. This approach also avoids duplication of efforts and ensures a common database to maintain configuration management.

Lesson: *Process architecture: Record process architecture using technology that supports iterative development and workflow.*

The SCAI DE/AE process is inherently iterative. The methodology used to record that process needs to be highly flexible. Project personnel have found that the $IDEF_0$ notation is not well suited for capturing such processes and at a minimum must be supplemented by work flow depictions. Although $IDEF_0$ can be

annotated with the required information, it alone cannot resolve the need for a high level graphic depiction of work flow.

Lesson: *Iteratively build up to megaprogramming.*

Apply an iterative build up approach when transitioning to a significantly new process. Better, faster, cheaper is a fantastic concept. However, when bringing in new technologies, sufficient time needs to be allocated up front to allow for training of personnel, responsibilities assignment, and proper planning of all activities to transition these concepts.

Lesson: *Avoid allowing documentation format to dictate the process.*

The Cleanroom Specification documentation format is very specific. If the Specification Team follows this format exactly, it may result in redundancies and inefficiencies in the specification documentation. Before specification writing starts, review the documentation format and decide upon a format that is most effective for the specific project. The SCAI team made significant alterations to the formats to adapt them to a process that is appropriate for the SCAI project.

Lesson: *Object oriented techniques complement Cleanroom engineering.*

Cleanroom Software Engineering provides a *rigorous* way of specifying and generating software to any desired degree of *statistical accuracy.* Booch Object Oriented Analysis provides an *informal* method for analyzing and abstracting a domain represented using "real world" concepts. Booch OO Analysis also provides a convenient notation for representing system differences and system commonalities (potential reuse) using inheritance and polymorphism. Several research efforts are underway to "fix" OO informal methods by combining the modeling process with a formal method. However, on SCAI, this union has already occurred.

2.4.4.3 Tool Support

Lesson: *Modeling tools should support behavioral abstractions.*

Information needs to be part of an object model to show when messages are passed from object to object. Because this was not part of the ROSE modeling tool, the information was maintained as Ada PDL separate from the model. Including this information would facilitate maintenance, consistency and configuration management.

Lesson: *Provide SEE support for debugging.*

One aspect to Cleanroom Software Development is formal refinement techniques emphasizing review and inspection. The implication is that unit testing is not necessary or desirable. As a result, minimal debugging support was included in the SCAI SEE. Finding an error in application code, infrastructure code, database code or the interfaces between them was a time consuming, expensive effort. Without debugging support, we were forced to break open code units, instrument the code, and re-compile to isolate the errors. Good diagnostic capabilities are a must regardless of the methods used during development.

2.4.5 Summary

The DE/AE teams have made significant strides on the SCAI Project, with many lessons learned from the preliminary domain analysis and subsequent development of the first megaprogramming release. We found that a domain architecture helps in managing engineering activities by providing a clear basis for organizing cooperating but "loosely coupled" groups of developers. We increased our appreciation of the need to integrate DE and AE processes in order to leverage the expertise of both disciplines and avoid the tendency for the two groups to drift in different technical directions.

We also gained first-hand knowledge of the synergy between domain-specific software reuse and demonstration-based architectural development. Demonstration-based software development (Ada Process Model) identifies architectural errors early in the development process before they become expensive to fix; domain-specific software reuse provides software components for assembling demonstrations that are progressively more accurate.

3.0 Process Support

3.1 Introduction

Prior to the STARS involvement, the SWSC had made significant strides in moving to an architecture based on open systems environment and developing practical approaches to generating reusable architectures and artifacts. They had made attempts to define high-level processes but had not progressed to define enactable details. As a result of using the SEI Capability Maturity Model (CMM) to assess the SWSC, the organization recognized the need to develop a detailed definition of their processes. The Deputy Assistant Secretary of Defense for Information Management had also chosen to sponsor a Corporate Information Management Workshop to define the processes performed in conducting software maintenance at the SWSC.

The SCAI project has created a Process Support Team which is responsible for ongoing process planning, process definition, automated process enactment, process tracking, process measurement and process improvement. This team supports Project Management, Domain Engineering, Application Engineering and other project activities. The team consists of:

(1) **Process engineers**, who develop, support and evolve the organization's process technology,

(2) **Process practitioners**, who are responsible for the process content, which is captured and executed with the support of process engineers and Software Engineering Environment (SEE) tools, and

(3) **SEE Engineers**, who provide, support and evolve SEE tools to support the above.

3.2 Long Term Objectives

The SCAI project outlined its long term objectives in the area of process as follows:

"Institute cooperating DE and AE processes including those for ongoing improvement and instantiate a process driven SEE to support the SWSC 'TO-BE' process."

Specific objectives include:

(1) Demonstrate process driven project planning.

(2) Improve a process as a result of being process driven:

- practitioners follow the process,
- process engineers are able to collect quantitative and qualitative metrics,
- metric data is analyzed, and
- process improvement is planned and implemented.

(3) Evolve the SCAI project into a process-driven organization.

Following the SCAI project, the intent is to institutionalize this approach to doing business throughout the SWSC.

3.3 Preparation Phase

3.3.1 Plans

Preparation Phase Objectives

During the Preparation Phase the SCAI project process objectives were to:

(1) Model and document the SWSC 'AS-IS' process and then define the SWSC 'TO-BE' process using $IDEF_0$ models and associated artifact definitions.

(2) Prepare the process support tooling for use in project planning and automation.

(3) Establish a method for formally defining and refining SCAI processes during the development phase of the project.

Planned Activities

The SCAI project contained several key processes which could not easily be represented as one process. The processes were:

(1) The SCAI Application Engineering Process[1] - the collective process to be defined to support developers to produce the SCAI-based space command and control system.

(2) The SCAI Domain Engineering Process - the process to be defined to support the development of:

- models characterizing the space domain,
- an architecture that satisfies a class of surveillance data processing applications that support both the domain subfamilies of space, air and missile warning, and
- reusable software components or applications to prepare pluggable software components.

(3) The SCAI Services Process - the process to be defined to provide support services, such as configuration management and software quality assurance for other SCAI processes.

(4) The SCAI Management Process - the process to be defined to support managers in the planning, delegation, dispatching, monitoring and controlling of processes and tasks on the SCAI project.

(5) The SWSC Process Improvement Process - the process to be defined to support the specification, design, development and improvement of SCAI processes.

Each of these processes satisfies a specific need. Each of these processes have defined interfaces and both request and receive information and services from other processes and as such can be viewed as peer processes. See Figure 7.

1. Subsequent refinement of the approach (beginning in the Preparation Phase and continuing to the present), caused the project to regard Application Engineering and Domain Engineering as an integrated process, since it was not effective to view them as separate. Please refer to Section 2.0, Domain Engineering/Application Engineering, for more information about the content of this integrated process.

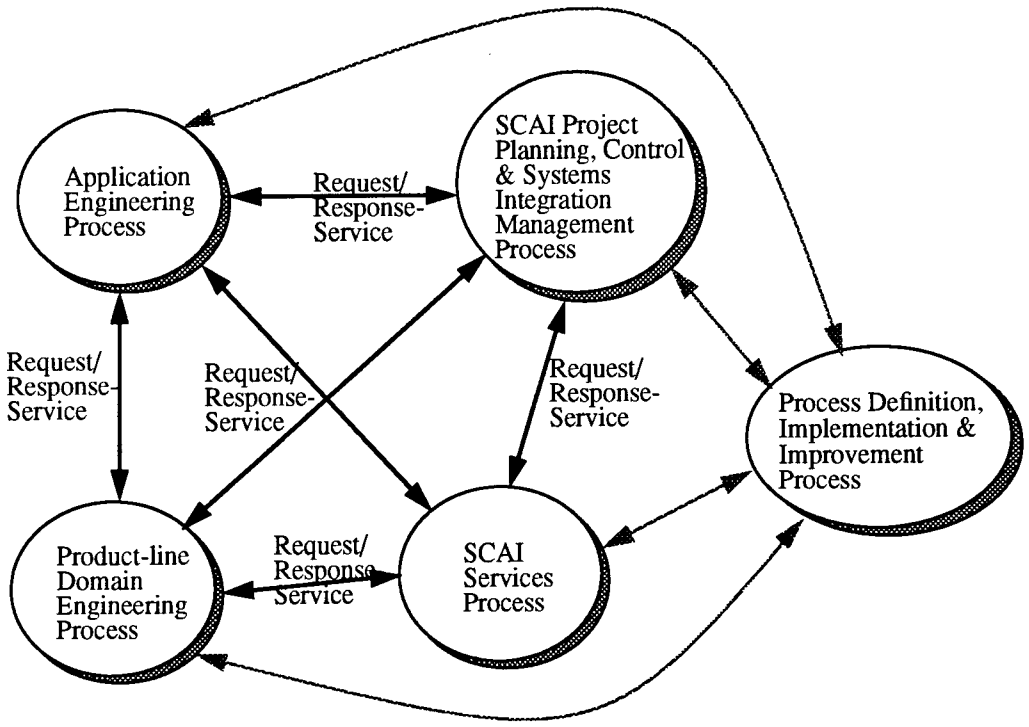


Figure 7. Process Architecture Concept

Each of the above processes is composed of process components and can be represented as an architecture where the interfaces of each process component are defined along with the service set that each process component performs.

This organizational concept supported our specification of the SCAI Application Engineering process and was used to identify the key interfaces between the SCAI Application Engineering process and the other SCAI processes.

Within the SCAI Application Engineering process there are a number of key processes that were identified. To support the planning for SCAI Process Definition, the steps identified in Figure 8 were performed. A strawman workflow of the SCAI project activities was defined from surveying the relevant existing processes, and identifying how the SCAI project was to meet its defined goals and its unique project requirements. This workflow analysis was prepared using the $IDEF_0$ activity modeling technique for representing abstractions of key project activities, while an ETVX view of the workflow provided basic control information for the process. SCAI contractors received briefings on both the $IDEF_0$ activity model and the ETVX workflow model and were involved in process reviews. The information collected from both views will be coalesced to form a SCAI AE process architecture from which an SPMS [1] based model of the SCAI process will be developed to support strawman SCAI project planning.

1. SPMS or Software Process Management System is the name used to identify the STARS-supported tool to support process specification and process-driven project planning and project monitoring. Its successor is the commercially-supported Process Engineering & Analysis Kernel System (PEAKS).

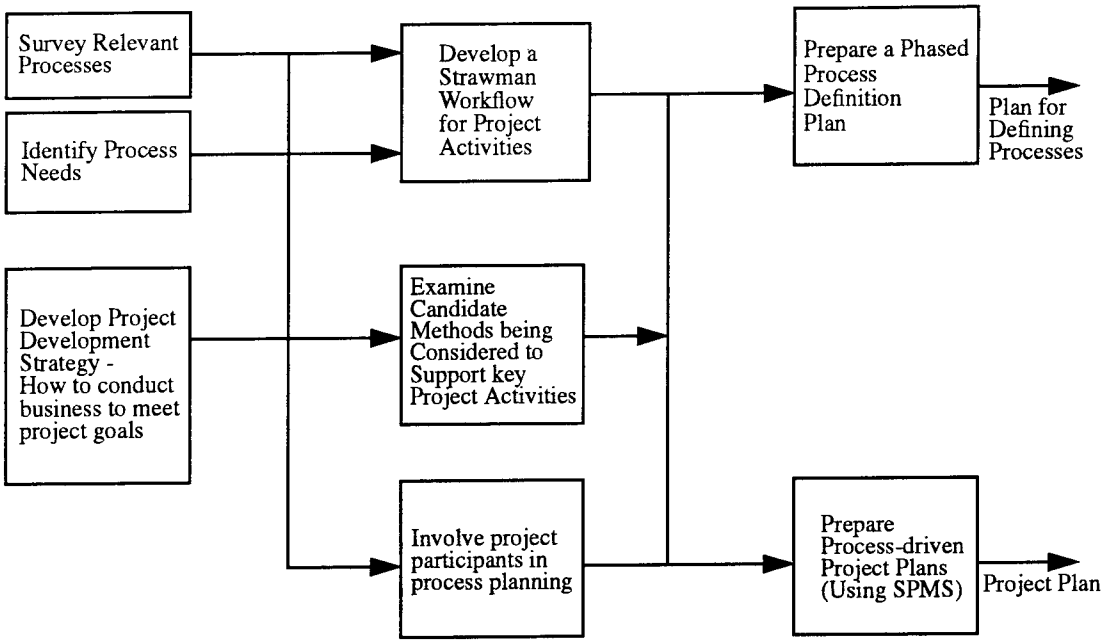


Figure 8. SCAI Process Planning Steps

The AE SCAI process architecture was used to prepare a phased process definition plan which was employed to determine the order in which SCAI processes would be defined using the Information Organizer Template[1] techniques for defining manually enactable processes. By performing the planning activities shown in Figure 8, the SCAI project will position itself to place key SCAI activities on a process-driven basis and provide the groundwork to actually plan portions of the SCAI project using the STARS concept of process-driven project planning.

Approach to Process-Driven Development

The SCAI approach to process-driven development began by first performing $IDEF_0$ activity modeling to describe the data flow between selected SWSC processes and to decompose these processes into their functional components. This resulted in a SWSC 'AS-IS' process definition. Aspects of this model were employed to prepare a proposed SCAI Process Architecture. This $IDEF_0$ depiction of the SCAI project process model will be augmented through the use of ETVX based modeling techniques to depict a more control oriented view of the process activities.

Using both process techniques to explain process concepts to the SCAI project personnel and management, the data collected from both views will be consolidated into the current SCAI Process Architecture which addresses both the architectures for the application and Domain Engineering processes. The $IDEF_0$ and the ETVX based techniques are both necessary to gain project consensus for the process. The $IDEF_0$ technique supports the concept of abstracting process components as process service objects and portrayed artifact flow between process service objects. The ETVX view will bring the $IDEF_0$ abstractions into focus by concentrating on describing activity precedence, process navigation, and exception handling. The resultant process work will provide the process architecture to support the SCAI project. This megaprogramming based system development process will also represent the SWSC 'TO-BE' process. These processes will be periodically refined, based both on data collected from SCAI project use, and from future SWSC project users.

1. See Appendix B for more information on these templates.

The overall SCAI process approach is shown in Figure 9.

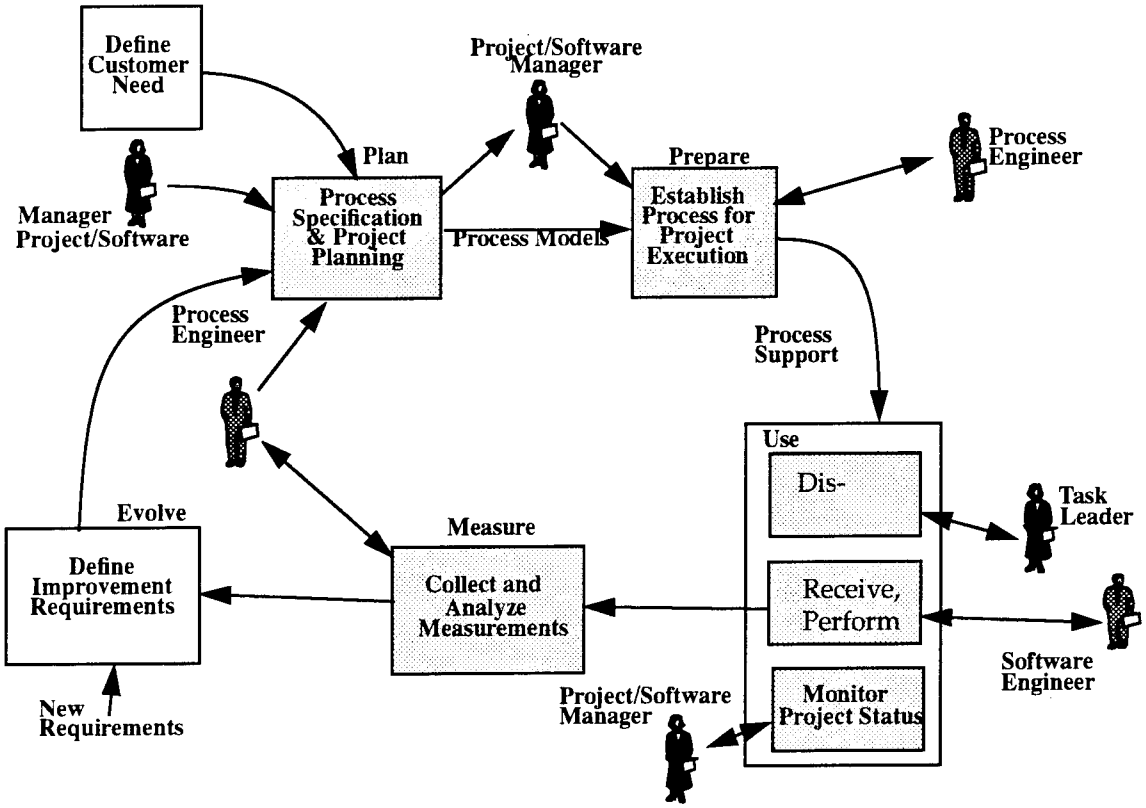


Figure 9. Process Cycle: Definition, Automation, Use, Improvement

Project/Product Context: The analysis and modeling of the SWSC 'AS-IS' process along with the unique requirements for the SCAI project will form the basis (or the requirements) for the SCAI process. During this phase, the process and project engineers will become familiar with the processes that are currently in use, as well as the new processes needed to support the RICC technologies and megaprogramming.

Plan: Based on the project and product context established in the above phase, a process specification will be developed using $IDEF_0$ models and Information Organizer Templates. These models and templates will in turn be used to develop detailed SPMS models. Project planning will them be performed by supplementing the SPMS models with resource, schedule, and artifact instance information. SPMS will then export all of this information into CAT Compass, the SCAI project management tool. This represents a key feature of the SCAI approach referred to as process-driven project planning.

Prepare: The above activities have established a well defined process and created a project plan for managing that process. The next step is to prepare for automated enactment of the process using the ProjectCatalyst and Process Weaver capabilities. This step is essentially the encoding and instrumentation of the process to be executed by the Process Weaver engine during project execution.

Use: The steps of the process and individual work tasks will be managed by Process Weaver throughout the SCAI project. In order to support a gradual transition to process driven development and to spread the considerable cost of the process definition and enactment, a phased plan was instituted.

During the Preparation Phase the SWSC 'AS-IS' process will be documented using $IDEF_0$. An overall $IDEF_0$ top level Process Architecture of the SCAI will be produced representing the framework of the SWSC 'TO-BE' process.

SCAI Release 1 will be performed using manual enactment supported by the $IDEF_0$ models and associated artifact definitions.

SCAI Release 2 will phase-in the use of the automated process planning and support tools. Process improvement will be performed based on the experience from Release 1. A formal process definition will be produced including detailed $IDEF_0$ and SPMS models as well as Information Organizer Templates. Automated metrics capture will be employed based on instrumentation embedded in the processes.

SCAI Release 3 will repeat the process as refined based on the measurements from the previous release.

Measurement: As noted above, measurements taken during the execution of each iteration through the process will be analyzed to determine how well the process performed. Amadeus will act as the primary measurement capture and measurement data base mechanism. SPMS and ProjectCatalyst will also support measurement capture and will provide the data to Amadeus for incorporation in the measurement data base.

Evolve: Based on the result of analysis from the previous step, a set of process improvement requirements will be defined. These requirements will then act as input to the next cycle of the process planning phase. As this cycle repeats, the SCAI process will continue to improve and by the end of the SCAI project will represent the 'TO-BE' process for future development with-in SWSC.

3.3.2 Summary of Accomplishments

The SCAI team accomplished the following during the Preparation Phase:

(1) Created the SWSC 'AS-IS' process model using $IDEF_0$,

(2) Created a SWSC 'TO-BE' process architecture using $IDEF_0$, and

(3) Developed the Information Organizer Templates for formally defining the SCAI process.

3.3.3 Analysis

Much time and effort was spent by project and process engineers learning about all the different methods and approaches being proposed for incorporation into the SCAI megaprogramming software development and maintenance process.

$IDEF_0$ was used to define the SCAI Process Architecture. The following steps were iterated to develop this model:

(1) Gather data by reading literature and interviewing experts.

(2) Bound the subject matter in a context diagram and define the purpose and viewpoint of the model.

(3) Structure activities and interfaces by gradually introducing more detail and refining elements in the model as more knowledge is gained.

(4) Present the model to experts and respond to their comments. [1]

The information in the architecture is examined over time as the model is used to conceptualize how the prevalent SCAI technologies fit together as a whole. Several iterations of the model were developed as the project explored processes such as MOIA, Cleanroom, OOA/D and the Ada Process Model with the accompanying technologies. The process architecture defines the Space and Warning System Engineering Meta-Context and the subprocess of the Space and Warning Systems Engineering processes, as well as the SCAI Process Engineering Context.

The SEI and STARS conducted a process definition class in December of 1993, focusing project attention on the criteria which are required to define enactable processes. Using the high level SCAI Process Architecture and the Information Organizer Templates project process, definers are expanding the definition of the project processes in the following areas:

1. *SCAI Process Architecture $IDEF_0$ Model*, Version 1.0 November 18, 1993, Prepared for: HQ AFSPACECOM Space and Warning Systems Directorate of Plans and Programs, Prepared By: SofTech, Inc. 985 Space Center Drive, Suite 320 Colorado Springs, CO 80915 (Currently CACI).

(1) System Engineering
    - Specification Preparation
    - Requirement Model Development
    - Architecture Model Development
(2) Incremental Software Development
    - Mission Application Development
    - Database Development
    - Display Development
    - Message Development
(3) Configuration Management
(4) Project Metrics
(5) System Certification

These processes will be generalized after the initial definitions are complete, and used to form a set of Domain Engineering processes.

3.3.4 Lessons Learned

Converging on Technical Approach

Lesson: *How technology is transitioned between organizations is as important as the technologies that are transitioned.*

Technology transition is at least N directional with N being dependent on the number of organizations that are advocating the use of a technology. When you transfer technology into an organization, both the transmitter and the receiver learn from each other. We spent a lot of time learning about technologies before we were able to integrate them into an approach. This was true for developing an approach to define our processes as well as for developing a process for Domain/Application Engineering.

Lesson: *When you plan to integrate multiple process definition technologies, any time spent by the team identifying and co-developing the process definition objectives is time well spent.*

Team members should also become familiar with all the technologies being considered or planned for use, and the functions each can or will provide.

Developing $IDEF_0$ Representation of the SCAI Process Architecture

Lesson: *$IDEF_0$ served as a good medium for representing SCAI Process Architecture.*

The Design/IDEF tool being used for SCAI works well for SCAI purposes of depicting the Process Architecture. Its primary strength is that it does a good job of graphically depicting the context of the SCAI engineering process relative to other SWSC processes that were modeled using $IDEF_0$ as part of the CIM initiative. Other modeling technologies are being used to supplement this process definition with information to facilitate process-driven planning, process enactment and measurement of the processes and the artifacts produced by them.

Lesson: *Process technologies can be combined to meet the differing needs or objectives of an organization. A single technology does not capture all the process information required.*

The IDEF methodology was not fully utilized by all process team members. This may be due to the fact that in the area of process definition, there were multiple technologies being proposed as appropriate for the SCAI project. No formal training in the alternative technologies was provided to all process team members early in the process definition lifecycle, so it took a while for the team to develop a strategy as to how the technologies would play together to create a process driven organization. IDEF gained wider acceptance near the completion of the process architecture development.

Lesson: *Review of the Process Architecture was painful.*

IDEF methodology calls for a Kit review where the model author provides reviewers with a drafted package, on which the reviewer is to place written comments, which the author then responds to. This package is passed back and forth until the model is complete and recommended for publication. We tried to use this approach but given constraints on team members' time, authors did not receive many comments. The authors also tried using unstructured interviews and walkthroughs to review the model.

Lesson: *People review process architecture when they are ready to use it.*

What we noticed was that people did not really critically review the Process Architecture until they actually started to use it to perform the process, or to develop the lower levels of process definition.

Lesson: *Review of high level process architecture by key people is critical to the minimization of rework.*

It is extremely important to ensure that close attention is paid to ensuring that the high level process architecture is reviewed in detail by key personnel. This is vital so that process definers can more efficiently and accurately expand the definition of project processes. Conduct the review of the high level process architecture in a round table discussion format. Simply passing out copies and soliciting feedback did not produce the necessary results. Although sometimes time-consuming, the round table review discussions proved much more effective.

Process Automation Technologies

Lesson: *Adoption of process automation technologies requires an incremental approach.*

Perform a pilot evaluation of process automation technology prior to operational use. Automated enactment of processes to drive project planning and the dispatching and tracking of a task's status is only very recently being prototyped. Manual enactment of defined processes has not really been tested and it is unclear whether this would be more effective than the automated dispatching of tasks and the automated tracking of task progress. Earlier prototyping should have been promoted even if it might have meant a certain amount of re-work due to evolving Process Support System (PSS) technologies.

3.4 Performance Phase

3.4.1 Plans

Performance Phase Objectives

During the first year of the Performance Phase, there were three major objectives:

(1) **Process Definition and Improvement Methodology**: Refine the SCAI process definition and process improvement methodology.

(2) **Process Content**: Elaborate the process definition based on the initial process architecture developed during the Preparation Phase, and improve the process based on usage experience.

(3) **Process Support Environment**: Use the STARS-sponsored Process Support Environment (PSE) to assist with process-driven project planning and process enactment, and provide usage feedback to improve the PSE.

Planned Activities

Process Definition and Improvement Methodology

(1) Complete the Process Definition Templates package - which specifies how to define and maintain a manually-enactable process (a joint STARS/SEI project) - and conduct user training.

(2) Provide usage-based feedback to help improve the Templates methodology.

Process Content

(1) Complete the decomposition of the $IDEF_0$ process architecture model to the subprocess level.

(2) Define key subprocesses using the above Process Definition Templates methodology.

(3) Model key subprocesses using ETVX[1] notation.

(4) Develop SCAI Engineering Database model to provide a basis for improving SEE integration.

(5) Define Metrics collection templates and imbed metric collection points into the enactment-level process definition.

(6) Manually enact key subprocesses as part of the SCAI Application Engineering effort and provide usage experience to improve the process.

Process Support Environment[2]

(1) Use SPMS (part of the STARS PSE) to transform the ETVX process model into a process driven-plan, and coordinate process-driven planning with the higher-level planning conducted by project management.

(2) Encode selected AE subprocesses in ProjectCatalyst (part of the PSE) and use ProjectCatalyst to manage and conduct day-to-day engineering work, starting with the SCAI Release 2 effort.

(3) Provide usage feedback to improve the PSE.

3.4.2 Summary of Accomplishments

Process Definition and Improvement Methodology

(1) Completed a draft of the Process Definition Templates package and began using the methodology.

(2) Provided usage experience for next generation of the templates package (in progress).

Process Content

(1) Refined $IDEF_0$ model of the Process Architecture and used it as a basis for detailed templates-based process guides for the following key subprocesses: Specification Preparation, Incremental Development, Certification, and Configuration Management; in addition, developed partial guides for Requirements Model Development, Architecture Model Development, and Project Metrics.

(2) Developed ETVX models of the Prepare Specification, Incremental Development and CM subprocesses.

(3) Manually enacted the Specification Preparation and Incremental Development subprocesses and refined them based on experience.

(4) Completed SCAI Engineering Database Design Report[3].

(5) Began work on Metrics instrumentation.

1. ETVX: Entry/Task/Validation/Exit - a workflow-based process modeling technique supported by SPMS (a component of the STARS PSE).
2. Please refer to Section 4.0, SEE Support, for a SEE perspective on experience with the PSE.
3. Refer to Section 4.0, SEE Support, for a brief discussion of SEE integration work that is in progress based on this analysis.

Process Support Environment[1]

(1) Used SPMS for the ETVX modeling, and gained experience in generating process-driven planning; provided usage feedback to the developer.

(2) Used ProjectCatalyst for SCAI Release 2 Specification effort, and gained experience in managing and enacting process-driven projects; provided feedback to the developer[2].

(3) Piloted and began use of a new tool, ProDAT, which improves the integrated support for IDEF process depictions (cutover of SCAI Process Architecture Model from earlier IDEF tool is in progress).

3.4.3 Analysis

In the final months of the prior phase (the Preparation Phase), Loral/STARS and the SEI collaborated to refine a joint process definition method for describing enactable processes. At the beginning of the Performance Phase, the Air Force decided to adopt the method - and in November, 1993, a class was delivered to the Demonstration Project process definition team. A plan was then put into place for defining SCAI processes based on the method taught in the class.

The method, termed the Process Definition Information Organizer Templates (PDIOT)[3] approach, was adapted for use by the SCAI team to take advantage of the $IDEF_0$ notation, which was considered by the Air Force to be fundamental to their process definition approach. IDEF was used to depict the "process architecture" in terms of a hierarchy of activities, connected by abstract inputs, outputs, controls, and mechanisms. Since the IDEF notation was not sufficient by itself to describe enactable processes (for example, activity sequencing cannot be shown using $IDEF_0$), the method called for supplementing the IDEF diagrams with textual and tabular data (via "templates") to describe the artifacts, the state changes of the artifacts and of the process as the process was carried out, methods to be used, agents to be coordinated with, etc. The supplementary textual and tabular data was to be documented via a conventional document production facility - in this case, FrameMaker.

The process definition team was broken up into subteams assigned to specific processes, with at least one practitioner assigned to each process (to act as the process "owner"), and scheduled dates were established for the completion of the initial process definition.

One of the earliest processes to be tackled was the Application Engineering "Prepare Specification" process, since quite a bit was already known about the specification process and since the project schedule called for the Release 2 specification activity to begin at around the time the team felt they could complete a detailed definition. The draft definition for this process was completed by March, 1994, and a pilot was launched to "try out" the process prior to its use for the Release 2 work. There were two simultaneous objectives for the pilot: to debug the process and to assess the viability of the STARS Process Support Environment (PSE) tools - SPMS and ProjectCatalyst - to support the Release 2 work.

Because of the focus on this particular process, good progress was made toward its definition - as well as its capture and support via the PSE. The Air Force decided to go ahead with plans to conduct the Release 2 specification activity using both the process and the PSE tools.

Meanwhile, other process definition teams were working on lower priority definition efforts, whose initial definitions (using the PDIOT approach) were completed incrementally during the first half of 1994. In all, four formal process definition packages were prepared during the first year of the Performance Phase - Prepare Specification, Incremental Development, Certification, and Configuration Management - and partial packages were prepared for several others.

---

1. Section 4.0, SEE Support, provides a more complete discussion of the Process Support Environment. In addition, a technical paper, "Using Process to Integrate SEEs", has been provided as an appendix to this report. This paper provides a theoretical model of a generic PSE and discusses how the Demonstration Project PSE maps to the theoretical model.
2. Refer to Section 4.0, SEE Support, for a discussion of the major PSE update (currently completing integration testing) that incorporates usage experience.
3. Documentation describing this method is in preparation for outside publication as of this writing; contact Mr. William Ett for preliminary information (ettb@lfs.loral.com, or 301-240-6322).

In retrospect, the process definition, process execution, and process improvement activities associated with the first process tackled - Prepare Specification - were relatively successful, and have resulted in a process definition package that is fairly close to actual practice. This success has not been matched for other processes, partly because there is less known about the other processes and partly because they haven't received the same degree of concerted attention by process engineers - complete with piloting activities.

Although the process definition, execution, and improvement techniques are steadily evolving, the following sequence represents the flow of activities that has been followed for each process:

(1) Define the process architecture (showing how the high-level activities interact within the process and with other processes),

(2) Develop the detailed enaction information,

(3) Model the process sequencing using an Entry/Task/Validation/Exit notation,

(4) Use the ETVX process to prepare an unscheduled plan which shows the integrated sequence of activities, validation points, and milestones needed to execute the process for one or more major project products (e.g., the three SCAI releases produced by the Incremental Development process),

(5) Schedule the process-driven plan and apply real project resource projections,

(6) Translate the scheduled plan into a WBS-oriented hierarchy of process-driven plan components, each of which will have an assigned point of responsibility,

(7) Execute the components by dispatching to practitioners the low-level activities called for by the process,

(8) Monitor the execution of the process via process-driven and product-driven metrics and make any needed ad hoc changes to the plan and/or process, and

(9) Evaluate the effectiveness of the process (taking into account the ad hoc changes cited above) and make adjustments to the defined process to improve it.

This full cycle has been realized in a repeatable fashion for the Prepare Specification process - albeit with a number of lessons learned along the way. Other processes have followed portions of the above cycle. Improvements have been realized for all processes, but formal adjustments to the defined process have lagged and will require later rework.

Several pragmatic problems with the supporting process technology can be cited:

(1) Volume

A great deal of detail is needed to define enactable processes. Developing all of this detail consumes much more time than can be predicted by groups that haven't done it before. In addition, having produced it, the effort to maintain it can be prohibitive for the organization. In retrospect, the volume of data produced for the formal process definition packages for the processes the team has achieved is non-maintainable, and is difficult for practitioners to use as part of an on-going process execution and improvement cycle. In response to this problem, a more streamlined approach is being developed which takes into account the needs for enactable definition information but is balanced against maintainability and usability objectives.

(2) Overlapping databases

Taking into account all of the forms in which process definition information has been developed for this project, there can be as many as five databases - maintained using different tools - that must be coordinated. These are:

- $IDEF_0$ diagrams and supporting information for the process architecture (maintained using MetaSoft's DesignIDEF tool - now being transferred to the ProDAT tool),
- PDIOT templates information for defining enactable processes (maintained using FrameMaker),

- ETVX models to support process-driven project planning (maintained using SPMS),
- Project scheduling and resource data (maintained using CAT/Compass and/or Microsoft Project), and
- Process execution support data (maintained using ProjectCatalyst).

Each database has an important role to play in supporting the process, but the number of databases compounds the maintainability problem cited above - despite a limited degree of integration among the tools.

(3) Tool support

In addition to the database problem cited above, each of the supporting tools had problems of its own - ranging from bugs, to usage paradigm problems, to interoperability problems. These problems are described in some detail in Section 4.0, SEE Support, and additional information is found in Appendix H, "Using Process to Integrate SEEs".

During this period, several steps were taken to improve tool support. First, at about the half-way point (July, 1994), it was decided to make a major upgrade to the STARS process support toolset - SPMS and ProjectCatalyst. The central change was to improve on the integration between the tools, so that minimal manual intervention would be needed to transition from the ETVX process definition data in SPMS to the coordination and execution data in ProjectCatalyst - thus lessening the impact of multiple databases. This upgrade was undergoing final acceptance testing at the conclusion of this period.

Second, in a parallel activity funded by the Sacramento/ALC EISE program, another group was working on an experimental prototype of an IDEF-centric approach to defining process (termed "criteria-based process modeling"). At the conclusion of this period, the Air Force decided to begin converting its IDEF data over to this tool, named ProDAT, in the hopes that its additional semantics will further enhance process definition automation.

Here are some of the additional steps being taken to adjust the approach, as of the end of the first year of the Performance Phase:

(1) More streamlined process definition views are being explored.

For example, the Application Engineering development team has devised a one-page process view which focuses on the walkthroughs present in the process. The walkthroughs represent validation points that gate the completion of milestones. This view has proven very effective in focusing the team on key aspects of the process and in providing management with a birds-eye view of status.

(2) Process support functionality is being absorbed into fewer tools - with commensurately fewer databases.

Notably, improvements to the SPMS tool are being accelerated so that it can handle most project planning activities and many process execution activities without outside tool involvement - thus consolidating three of the aforementioned five databases into one.

(3) More advanced process support automation is being explored.

In addition to the above improvements to SPMS, the Air Force is hopeful that the ProDAT tool will help by consolidating the IDEF and FrameMaker data into a single tool which supplies considerably more expressive power and semantic checking support for the models.

Thus, the project is engaged in working out an adjusted approach to process definition and improvement. In fact, it seems reasonable to predict that any organization with advanced process ambitions (such as a megaprogramming organization) will be engaged in improving its process technologies throughout the life of the product-line.

3.4.4 Lessons Learned

Lesson: *The initial overhead associated with process definition is significant.*

There has been a significant effort associated with almost every aspect of process definition for the SCAI project, which has taken the project more time than was originally anticipated. Approximately 17% of the total project effort expended over a two year period has been consumed in some form of defining processes. This unanticipated overhead is in part due to the fact that people tend to think that they understand their process well enough to document it, and that they are capable of capturing it correctly and usefully. People underestimate the amount of effort required to transform their abstract notions into concrete enactable formats as well as the complexity of the resulting data. This transformation of process from the abstract to the concrete was further complicated on the SCAI project by the number of organizations contributing to the definition of the SCAI DE/AE process. Even slight variations and interpretations in artifact definitions can perturb the process definition schedule.

Lesson: *Process definitions need to specify how to administer the relationships within and between processes of the project, especially when different organizations are involved in performing on the project.*

The SCAI process definition effort defined the criteria for quality software engineering products and the activities that needed to be performed to develop megaprogramming DE/AE system engineering products. What the SCAI DE/AE processes failed to define however, were the rules governing the relationships between the roles of individuals performing different aspects of the DE/AE processes, and between the multiple organizations involved in development of the SCAI products. As a result of the hybrid nature of the DE/AE process, different organizations were allowed to use different methodologies to perform their part of the SCAI process. When these organizations were working within different layers of the SCAI architecture this caused no problems, however, when these same organizations used different methodologies working within the same layer of the architecture the AE team lead found herself spending an inordinate amount of time (estimated to be three times the amount of time that it would have taken to manage a single contractor) coordinating the integration of system components and solving issues that arose as a result of the differences between organizational cultures.

Lesson: *Becoming process driven has payoffs.*

People have processes by which they are accustomed to getting things done which for the most part are not explicitly stated. When you introduce the concept of examining and explicitly defining process there will be resistance. This will be especially true when the same people are also expected to produce their product using the process at the same time you are asking them to define it. However, our lead astro-physics engineer is now a proponent of process definition. Initially he was very resistant to defining processes because his inability to "free-wheel" the design and development of code was causing him schedule slips. Once the process was defined he discovered that people with no background in astro-physics were able to follow it and produce code that provided analysis of system data with the same accuracy as the current system with little to no debugging of their code on his part. His old way of doing things would have cost him hours of time debugging code.

Lesson: *Maintaining detailed process information is unwieldy without integrated automated support.*

The Process Definition Information Organizer Templates were primarily designed to facilitate the collection of information needed to define an enactable process. A process engineer would then organize that information to build either manual or automated enactable process models. On the SCAI project these templates were initially used in conjunction with $IDEF_0$ process models to create Process Guides (a manually enactable process definition). At the outset the level of detail required to specify a process was not well appreciated. This resulted in a large number of templates being developed using a word processor. Handling all the files and ensuring consistency among the various templates and the IDEF model became very time consuming. In order to avoid jeopardizing the integrity of the Process Guides major modification to either the IDEF models or the templates has been avoided. The project is now starting use of the ProDAT tool to assist with this front-end process definition burden.

Adding to the burden of the front-end process information problem, the project is also using the STARS SPMS tool to model ETVX depictions of the processes and the ProjectCatalyst tool to present enactable renditions of the process to task leads and practitioners. The new version of these PSE tools are much better integrated, which should improve the situation substantially.

In general, the project views process data integration as a pressing long-term requirement.

Lesson: *Simplified views of process are required by those performing manual enactment.*

As mentioned above, a significant amount of effort was expended to define the process in sufficient detail to support the megaprogramming concept of "process-driven". The products developed ($IDEF_0$ models, STARS/SEI/SCAI Process Definition Information Organizer Templates, and SPMS models), were focused on providing information for the process engineer. Detailed process guides, while useful to bring a novice on board, have proven too large for day-to-day guidance.

In one case AE development process practitioners have abstracted the manually enactable process guide into a one page state transition diagram, backed by a single page description for each state. This approach was taken to keep the diagram simple so that it could be represented to the team that had to implement it in a simple and understandable way.

Lesson: *$IDEF_0$ is easily misunderstood by process enactors.*

IDEF was designed to give people the capability to build logical models that display the semantic characteristics (meaning) of business transaction. It was intended to depict the functions of a business, the relationships between functions and the information used by those functions. The IDEF semantics use hierarchy, and left to right positioning on a page, but in describing business functionality it was never intended to indicate sequencing of events. Processes or activities that are being enacted, however, must provide information about both importance of activities, and timing of them. Positioning of activities in IDEF models was often misunderstood by people who were trying to follow them. Additionally, the bundling of "ICOMs" (Inputs/Controls/Outputs/Mechanisms) obscures the data flows that occur at lower levels of the models.

Lesson: *An incremental application development strategy provides an excellent vehicle for incremental process development and improvement.*

Lesson: *In conjunction with the incremental buildup of the organization's process and its development of supporting process technologies, pilot activities at key project transition points are essential.*

The initial definition of the DE/AE process was captured concurrently with performing the process for the first SCAI release (i.e., first production of Specifications and Release 1 products). During the second release activity, the process was refined and improved based on experience of the first release. For the third release, we expect to take further advantage of our experience to significantly increase our efficiency and effectiveness. At key junctures, before attempting to transition to a new approach, the project conducted pilots - simulating how work would proceed using the new approach. This has proven very effective at introducing the new ideas and techniques to the users and to "shake down" any last minute problems.

Lesson: *An organization seeking to overhaul their processes should opt for process support technologies that are proven and stable.*

In general, this organization has sought to make significant changes along multiple technology fronts at once - this was to some degree inherent to its charter. With respect to process, in particular, we sought to develop new processes (integrated under a new process architecture), use a new way of recording process definition, and use a new process support toolset. Each of these technologies would be a significant challenge: the combination of the three has been exceptionally challenging. The Process Definition Templates approach was developed in conjunction with the SEI and its first use was on the SCAI project - in essence, a "Beta test" of the process definition approach. In fact (as discussed in other lessons in this section), actual usage has disclosed pragmatic problems with the approach, and this experience is being taken into account for a revised approach, now under development. The process tools approach was similarly ambitious, and its second generation - taking advantage of usage feedback - is nearing completion of integration testing as of the time of this report. The Templates approach was introduced to the project at the start of the Performance Phase, and the Process Support Environment was introduced about four months into the Performance Phase. In both cases, the new process technologies are viewed as strategic to the SWSC product-line approach. But in both cases, there has been significant impact to the process definition and enactment activities themselves.

Lesson: *A technical leader, such as a process architect, with overall decision responsibility is required to keep the project effort focused.*

Given the technical and organizational challenges presented in the foregoing discussion, it is clear that any project with similar ambitions must rely on a focal point individual for process engineering as a whole: someone charged with coordinating process definition and the technologies used to define, enact, and improve the process. For the SCAI project, an Air Force officer filled this role; and with the cooperation of the team, the project has made substantial progress in building up process understanding as well as tuning and adapting the new process support technologies.

3.4.5 Summary

Formal process definitions - including process models, and enactment information templates - have been developed for several SCAI processes, and to varying extents full process execution and improvement cycles have been realized. The cost of formal process definition is significant, particularly for an organization with little prior experience in conducting such activities. We found that close to 20% of the effort on the Demonstration Project went into process definition. In retrospect, the initial level of detail attempted for defining these processes was excessive - particularly given the limitations of the supporting toolset - and the project is currently exploring improvements to streamline the approach, reduce the number of tools (and associated databases), and upgrade the capabilities of the remaining tools.

Based on the experience of defining and using these processes, we believe that a project needs a technical leader responsible for process architecture and that the process should be developed iteratively.

4.0 SEE Support

4.1 Introduction

Megaprogramming, according to STARS, requires a significant level of SEE support to provide automated assistance to the product-line's process definition and enactment.

During the Preparation Phase, the Loral/STARS team acquired two major increments of SEE hardware and software and added it to the existing SMX equipment for the Demonstration Project. Since the project was simultaneously developing details of its overall approach, early decisions had to be made about SEE composition prior to availability of detailed understanding of the process the SEE was to support. Key decisions were to use IBM workstations, Rational software (for Ada-oriented software engineering), and TRW software to provide the architectural infrastructure support. In addition, because of the major emphasis on process, the decision was made to proceed with development of the STARS-sponsored Process Support Environment (PSE), designed to provide advanced process modeling and enactment capability. The SEE was used by end-users during the Preparation Phase to develop system and software engineering models to capture domain understanding and lay the groundwork for the SCAI specification effort.

During the first year of the Performance Phase, the team planned and ordered the third and final major SEE increment - but due to STARS funding profile alterations, the increment's installation was delayed significantly, and an installation originally planned for April, 1994 was just beginning installation in October, 1994. This delay degraded the team's productivity, although remedial work pattern adjustments helped to offset this impact. The STARS PSE, delivered later than originally anticipated, was used during the SCAI Release 2 work, and usage experience was used to define a major update, which has begun pilot/acceptance testing at the time of this report. SEE integration initiatives took place during this period in the areas of process/SEE integration, configuration management, documentation production, metrics, and overall engineering database analysis. Although production work for the first two of three SCAI releases took place, the team was making substantial adjustments to their approach - necessitating similar approach adjustments in SEE strategy.

As the project enters the second (final) year of the Performance Phase, a recent end-user survey indicates that most users are satisfied with the functionality of the SEE (except for SEE performance, as mentioned above), and that they feel the SEE is headed in the right direction to support the maturing product-line strategy.

4.2 Long Term Objectives

The requirement for the SCAI SEE environment is to create a framework populated with tools, supported by well defined processes and procedures, tailored as necessary to the needs of the particular application domain. In addition, the environment shall provide well defined software engineering, project management and lifecycle control procedures, processes and methods.

4.3 Preparation Phase

4.3.1 Plans

Preparation Phase Objectives

To prepare to address the long-term objectives, the major Preparation Phase objectives were to:

(1) Acquire and install the first increments of SEE hardware and software and integrate the new assets into the existing SWSC/SMX SEE network, and

(2) Formulate a joint SEE integration strategy with the Air Force and begin the integration work in preparation for the Performance Phase.

The intent of the Preparation Phase was to build up sufficient SEE capability to support the project's initial Performance Phase activities.

Assumptions/Constraints

Certain givens were assumed at the outset of the project.

(1) The products in the SEE must:

- Follow open integration standards, and
- Be available on several hardware platforms.

(2) The SCAI SEE must support all members of the SCAI development team, which was initially planned to be about 20 engineers. Adding management, support, and Loral STARS personnel would increase the number of users to over 30.

(3) The application will include about 200 KSLOC of Ada[1], with major portions of the application being reused from existing work.

(4) The SCAI SEE must support the following process areas:

- Product-line management,
- Domain Engineering,
- Application Engineering, and
- Application target testing.

(5) Some use must be made of AF residual assets, including hardware, software licenses, and network elements[2].

(6) The majority of the SCAI SEE elements must be commercially available products.

(7) SEE planning must accommodate several new SEE elements that are products under development or in beta test. This is especially applicable to products coming from STARS technology development efforts.

(8) The SEE must be integrated to the highest level possible, based on the capabilities of the underlying integration mechanisms, and the intended use of the tools on the Demonstration Project.

(9) The creation of the SCAI SEEs must be constructed in the presence of several pragmatic constraints.

- Budget - The FY 1993 budget for the SCAI SEE was set at $500,000, with acquisition starting in January 1993 and completed by July 1993. The objective was to spend about half of this budget on hardware and about half on software. FY 1994 will require a similar budget, with a plan of spending 30% on hardware, 40% on software and services, and 30% on training[3]. Subsequent year budgets will have to address upgrades, maintenance, and potential staff growth.
- Security - A portion of the project artifacts will be DoD classified, requiring a segment of the SEE to be classified with no outbound external communications. This security situation has had a significant effect on the connectivity to project team members working outside the main SCAI project site.
- Work Locations - Some members of the SCAI engineering team will be located in separate contractor facilities and do not have access to the SCAI SEE.

1. This estimate of application size turned out to be low by at least a factor of two.
2. *Demonstration Project Baseline Report*, Loral STARS CDRL No. 5052, April 1993.
3. *SEE Integration Plan*, Loral STARS CDRL No. 5200, June 1993.

- SEE Product Development - Some of the elements of the SCAI SEE are not fully developed and are not yet commercially available. (SPMS and ProjectCatalyst are examples.)

Planned Activities

Loral, as the STARS prime contractor paired with the Air Force, was to take on the role of coordinating the initial acquisition for the Preparation Phase. The intent was to transition this capability to the Air Force for increments following the Preparation Phase. Responsibility for acquiring and installing some incumbent products (including the Rational Ada support environment, UNAS/SALE, etc.), rested with the Air Force. The Air Force was to retain installation and system management responsibility, in keeping with their existing role for other SEE assets.

A joint Air Force/Loral SEE working group was to be established for coordinating the team's SEE needs and developing solutions and schedules. The overall SEE integration strategy was to be tied to the SCAI process; and since the process was being developed during the Preparation Phase, coordination with the process support group was required.

Loral was to continue development and begin installation and training for the STARS-sponsored process support toolset.

4.3.2 Summary of Accomplishments

(1) Acquired and installed initial increments of SEE hardware and software,

(2) Developed initial overall SEE integration strategy, documented in Version 2.0 of the *SCAI Demonstration Project Management Plan* (SDPMP),

(3) Enhanced RICC architectural infrastructure support software (refer to Appendix B for details) and continued movement towards commercialization, and

(4) Continued development of STARS-sponsored process support toolset, including integration work to establish a coherent process support system; conducted early pilot work with SPMS, one of the key tools in the toolset.

4.3.3 Analysis

Early in 1993, a joint team of Loral STARS and AF engineers was established to assemble and integrate the SCAI SEE. Plans and schedules were established. This effort was described in a series of documents during 1993, which detail the accomplishments and changes in the plan[1].

In creating the SCAI SEE, the joint team acted as a general contractor. They completed selection and installation in stages, as decisions were made, and products arrived. The team members were the consensus gatherers and purchasing agents.

The AF team members were the installers and initial users. The team was supported by several product vendors.

While the team made significant progress against the plan, several activities were not completed or reached only limited success. In general, most activities were far more time consuming and complicated than initially envisioned. There were many factors and options that needed consensus decisions. Many of the issues were unavoidable, and there are no recommendations on how to eliminate them, except to plan more labor. The following is a list of some of the issues that impacted the results of the establishment of the SCAI SEE:

1. *SEE Integration Plan*, Loral STARS CDRL No. 5200, June 1993. *Demonstration Project Management Plan*, Version 1.1, Loral STARS CDRL No. 5050, June 1993. *SCAI SEE Description*, Loral STARS CDRL No. A010, October 1993, and the *AF/STARS Demonstration Project Management Plan, Version 3.0*, Loral STARS CDRL No. A009, November 1994.

(1) The AF and their contractors had an installed base of software and experienced users in that base. The selection of the SCAI SEE products required significant consensus efforts to incorporate this existing base.

(2) The joint Loral/AF team worked in different locations and different time zones. Phone line and electronic communications (E-Mail and FTP) into Peterson AFB were limited. This situation placed a stress on normal communications between team members.

(3) The SEE included products from many vendors, integrated through several open system standard interfaces.

(4) Several of the products (e.g., process support tools) had release and development delays. The products were less mature than expected.

(5) Products and prices changed during the selection and acquisition period. Budgets and plans required several adjustments.

(6) The SEE was distributed among Loral, the AF, and AF contractors in several locations. This issue required effort to establish network capabilities and diverted attention from integration efforts.

(7) A major portion of the SEE is classified, and without connections off the AF base. This was not in the original scope of the STARS Demonstration Project plan. This issue limited support from people outside the classified SEE.

(8) Less than half the planned funding was available in 1993. This delay in funding left the acquisition of classes of products to 1994, and reduced the scope and function of the SCAI SEE in 1993.

SEE Strategy Development Experience

The effective use of megaprogramming concepts requires advanced technology support. Methods and tools are required to achieve compatibility of data, control, and processes, directed at developing shared assumptions about the system being developed.[1]

The long term requirements for the Demonstration Project SEE are stated in the "STARS Follow-on Contract". [2] The following paragraphs provide the relevant extracts of that contract:

> "In general terms, the environment shall consist of a framework populated with tools, supported by well defined processes and procedures tailored, as necessary, to the needs of the particular application domain. In addition, the environment shall provide well defined software engineering, project management and lifecycle control procedures, processes and methods.
>
> The contractor (Loral) shall develop and deliver to the Demonstration Project team, a product-quality, multiple seat software engineering environment tailored to the specific application domain in accordance with the requirements of the Demonstration Project. The environment will serve as the vehicle for developing and demonstrating this technology and for supporting new software engineering procedures, processes, and methods.
>
> The contractor (Loral) shall, using the results of previous STARS activities, install and demonstrate the software engineering environment tailored to the specific domain of the Demonstration Project."

The SCAI SEE strategy to meet the SEE requirements is based on the following set of variables:

(1) connectivity,

(2) number of users,

(3) user roles, responsibilities, and tasks,

(4) SCAI team process definitions,

---

1. Boehm, B. W., *Megaprogramming*, Preliminary Version, April 1991.
2. *STARS Follow-on Contract*, F19628-93-C-0129, ESC USAF, Hanscom AFB, MA, August 1993.

(5) artifact sharing,

(6) user experience,

(7) cost/benefit of integration effort, and

(8) technology transition value to SWSC support team.

Integration of the SCAI SEE is grouped into four areas, the Presentation Integration, Process/Control Integration, Tools, and Data Integration. Cap Gemini's Process Weaver, IBM's WorkBench, and standard relational databases are the primary mechanisms of integration in the SCAI SEE. Figure 10 illustrates the integration areas and components.

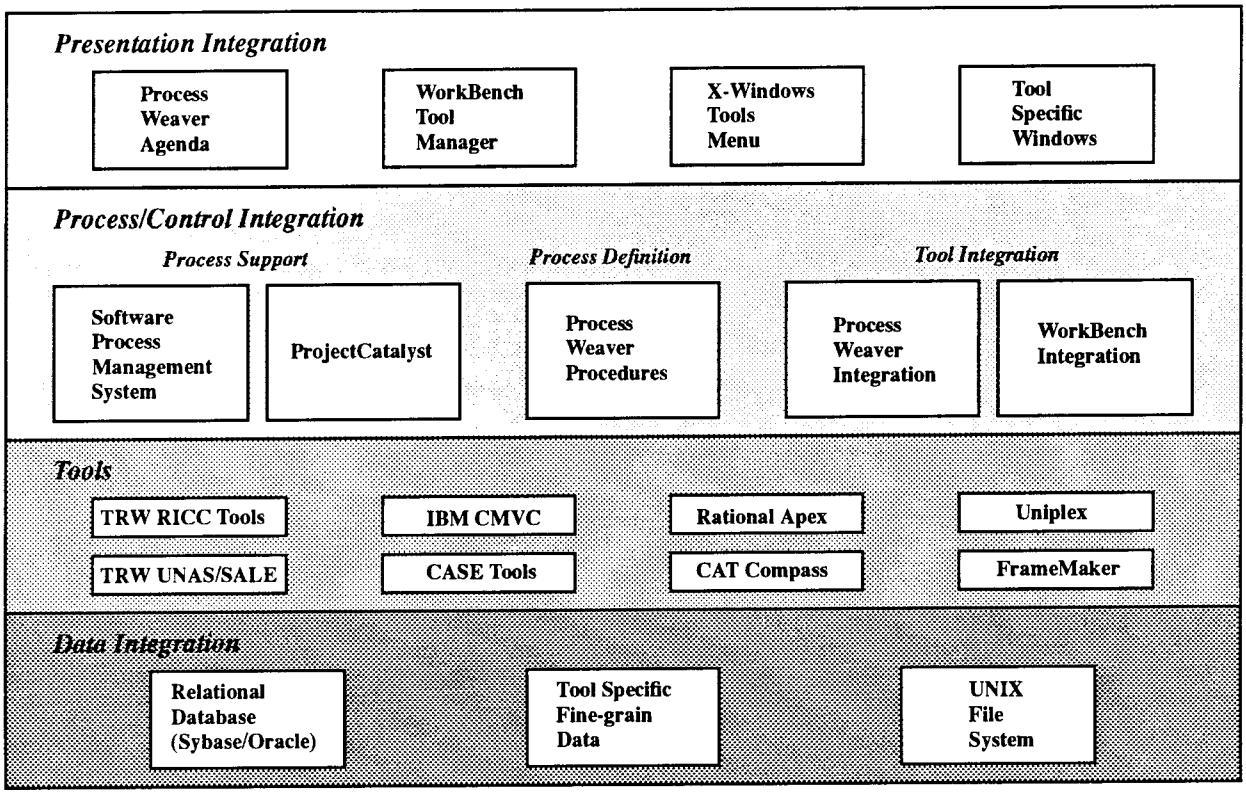


Figure 10. Integration Areas

The main feature of the Presentation Integration area is the project process guidance provided by the Process Weaver Agenda and Work Context windows. The WorkBench Tool Manager provides an interface to tools outside of project processes, such as mail. Tools are invoked through Process Weaver and WorkBench, and display their operation specific windows. The X Windows tools menu is used to start Process Weaver and WorkBench.

The Control/Process Integration area includes the Process Weaver and WorkBench integration mechanisms. Process Weaver procedures for the defined project processes provide the majority of process guidance in the SCAI SEE. The STARS developed Software Process Management System (SPMS) and ProjectCatalyst help model, plan, and define the project processes to generate the Process Weaver procedures.

The Tools area includes tools selected by the SCAI team, based on their currently installed products and the skills of the engineers.

The Data Integration area is supported by commercial relational databases, the operating system file system, and tool specific fine-grained data storage.

SEE Process Experience

To meet these goals, within the constraints and assumptions, an iterative and incremental approach was followed. The following is a list of the major elements of the process and the activities within each element. The observations, analysis, and recommendations in this report are against this approach.

(1) Select SEE Products: research products, gather requirements, define criteria for selection and publicize results of selection.

(2) Acquire SEE Products: develop preliminary plan for acquisition and use, justify cost to management, initiate acquisition, provide technical support for acquisition and verify receipt of products.

(3) Assemble SEE Products: install products, run sanity tests on products and adjust products for local installation specifics.

(4) Integrate SEE Products into a SCAI SEE: develop experience in using product, integrate products using process, presentation, control, and data integration techniques, publish administrator guides and publish user guides.

(5) Deploy the SCAI SEE: demonstrate the SEE to users and release the SEE for general use.

(6) Revise and maintain the SCAI SEE: collect problem reports, respond to reports; collect change requests, operate change board and survey users.

4.3.4 Lessons Learned

Lesson: *Ideally define the organizational processes prior to selecting and acquiring the SEE toolset.*

The selection or acquisition approach should not be started until you have a general definition of the scope of the engineering effort and the processes that the engineers will follow. The recommendation is to not assign anyone on a proposal team to SEE product definition until the software engineering process has a high level definition and until the SEE product evaluation process is defined and documented. It is estimated that, because of the lead time and cost needed to acquire SEE products, most projects define a toolset early in the proposal, and get the SEE products locked into the contract cost. The definition of processes is viewed as something that can be done after the contract award. Unfortunately, this leads to parts of the process with no automation support, or products being acquired without a process to use them (shelfware).

Lesson: *The agreements and licenses for acquisition and maintenance differ for each product.*

This situation creates variation in levels of support and installation flexibility. For some products, there is sufficient immediate support, while for others there is virtually no support. Some products are bound to specific system configurations, and it is time consuming and sometimes costly to go to the vendor for adjustments.

Lesson: *It was difficult to plan when a product would arrive or how it would integrate into the SEE.*

The variability of products being delivered to the SCAI SEE assembly site requires frequent changes to schedules and plans. When trying to integrate several products, the delay in one product will delay the whole integration. Equally serious is the lack or limitation of intended function due to product development problems. Usually adjustments can be made for these problems because of the variety of capabilities in the other products and the use of manual processes, but disruptions in the plans will occur.

Lesson: *Upgrades to an operating system (or other SEE infrastructure components) can disrupt integration.*

In an integrated SEE, many of the tools will depend on particular versions of other tools (or parts of the SEE), so that upgrades cannot take place in isolation. Rather, upgrades have to be coordinated with one another. Infrastructure tools (such as operating systems or data bases) have many dependencies, so special care must be taken when planning an upgrade to them. Some upgrades may have to be delayed until dependent tools can be upgraded. Code may even have to be written to temporarily mitigate the effect of upgrades. We recommend building a tool dependency matrix, to better understand the impact of proposed upgrades.

Observation: *Integration complexity increases as the number of tools/vendors increases.*

There are clearly advantages and disadvantages to having a SEE which requires integration of products from multiple vendors. The SEE team is in the process of distinguishing these factors and is trying to determine how the factors impact integration complexity.

Lesson: *Plan an incremental approach to SEE integration.*

In view of the difficulties in adequately defining the organization's processes in time and in view of difficulties in planning integration because of fluctuating delivery schedules and complex dependencies, we recommend adopting an incremental approach to SEE integration.

Lesson: Allow for training of SEE integrators.

It is likely that unless the organization is exceptionally well prepared, there will be experience deficits in some areas. To remedy the deficits, plan for concentrated training of the integrators or hiring of product experts. Consider selecting products and techniques that the SEE integrators know.

Lesson: *Plan sufficient labor for SEE acquisition and assembly.*

When selecting, acquiring, and assembling a SEE for a small software engineering team (about 20 engineers), plan about 16 labor months for the effort, with a minimum elapsed time of about six months. This does not include SEE integration, deployment, or support efforts. If delays occur in this effort, as experienced on the AF/STARS Demonstration Project, additional labor will be required. The AF/STARS Demonstration Project spent an estimated 14 labor months on these activities during 1993. (During 1994, to complete these activities, the AF/STARS Demonstration Project plans to spend another eight labor months.)

4.4 Performance Phase

During this period, substantial progress was made towards an integrated SEE to support megaprogramming - with a recent user survey characterizing the SEE as providing adequate support and headed in the right direction - but more work is needed to improve functionality, performance, and integration.

4.4.1 Plans

Objectives

The project had the following major SEE objectives during this period (10/93 - 10/94):

(1) Apply the SEE that was built up during the Preparation Phase, assess its use in supporting SCAI engineering work, and evolve the SEE strategy based on initial experience.

(2) Install and begin use of the STARS-sponsored Process Support Environment (PSE)[1].

(3) Address SEE architecture and process issues associated with government security and the geographical dispersal of the team.

(4) Acquire, install and begin use of the final major increment of SEE hardware and software.

(5) Improve SEE integration in the target areas of process support, configuration management, documentation production, and metrics.

(6) Refine the SEE improvement process.

(7) In general, continue to lay the groundwork for a product-line SEE to support future SWSC applications.

1. The PSE includes SPMS, a project modeling and process-driven planning tool from ccPE, and ProjectCatalyst, a process management and enactment tool from SET. ccPE and SET are subcontractors of Loral FS.

Assumptions

Please refer to the Preparation Phase assumptions documented in Section 4.3.1, Plans; these assumptions continue to apply, with the following updates and additions:

(1) Number of Seats - increased from 30 to 50.

(2) Size of Application - increased from 200 KSLOC to over 400 KSLOC.

(3) SEE Acquisition Budget: FY 1994 STARS funding decreased from $750K to $500K, and the funding was delayed from 12/93 to 5/94 (this is elaborated below).

Planned Activities

The SEE plans for the first year of the Performance Phase were:

(1) Increment 3 of the SEE (the final increment of hardware & COTS software) was planned for installation in the 4/94 - 5/94 timeframe. The acquisition was significantly delayed, however, and was not ordered until 8/94 - and installation is not yet complete. The delay was caused by:

- STARS funding reallocation, leading to a SEE budget reduction from $750K to $500K, and delaying funding from 12/93 to 5/94. This change caused two cascading effects, which delayed the ultimate installation seven months:
  - Loral personnel were unable to handle the acquisition as planned, due to scheduled personnel reductions; the Air Force assumed the responsibility, but there were impacts due to the unexpected staffing requirements, and
  - The acquisition was shifted into the latter part of the fiscal year, which increased the Government acquisition window from a planned 3-5 months to 5-6 months.
- Fluctuating requirements: number of seats, location of personnel, etc., necessitated several rewrites of the requisition.

(2) A major release of the STARS PSE was planned for January 1994, to be followed by a two month pilot, and followed in turn by production use for the SCAI Release 2 specification effort. This activity actually started in March, due to delays in the completion of the PSE.

(3) A classified T1 link was planned to be installed at each of the two primary engineering contractor sites, thereby extending the SEE to include those personnel and SEE assets. At the start of the period, these personnel were forced to use segregated, unclassified SEEs, which impacted their efficiency. Only one T1 link was acquired, and the other group was relocated to reside on-premises at the SWSC.

(4) Provide a high-speed Internet connection from the unclassified SEE assets to allow off-site personnel to provide efficient support and to facilitate general project communication with the outside world. This was not accomplished during this period.

4.4.2 Summary of Accomplishments

Table 6 itemizes significant accomplishments with respect to Demonstration Project SEE support. The discussion is divided into the same five categories that are used in succeeding subsections.

SEE Management	• Connected all project personnel to single classified SEE • Acquired and applied interim SEE assets, to offset impact of delayed SEE Increment 3 • Ordered SEE Increment 3
SEE Engineering	• Piloted and adopted several new tools (IBM CMVC, STARS PSE, SoDA, ProDAT)[a] • Completed and delivered first release of STARS PSE; gathered usage experience and scheduled major PSE upgrade (now completing final integration testing) • Completed prototype integration between IBM CMVC and Rational CMVC • Completed artifact database study, laying groundwork for future SEE data integration
Megaprogramming End-User Functionality	• Overall: End-user survey showed that SEE functionality is adequate and headed in right direction • Workgroup Efficiency: - All users now on same SEE - Instituted use of IBM CMVC for general purpose tracking (problems, action items, etc.) • Process Support: - Applied STARS PSE to SCAI Release 2; provided usage experience leading to major PSE upgrade - Began use of the ESIP[b]-sponsored ProDAT tool to support SEI/STARS process definition • Metrics Support: Using Amadeus for labor metrics collection • Domain/Application Engineering Support: Applying state-of-the art tools for SCAI engineering work, including: - Rational Rose (object-oriented domain/application modeling) - RICC Tools (architectural infrastructure development) - Rational Apex (Ada design/development)
SEE Improvement Process	• Conducted two user surveys (in December, 1993 and August, 1994) and developed improvements to survey techniques • Automated the SEE system management work order process using IBM CMVC
SEE Technology Transition	• Conducted presentations and demonstrations of the SEE to the SWSC and to outside organizations • Conducted classes and pilots to introduce new toolsets into use • Actively participated in the SWSC Tools Integrated Process Team (part of SEPG) • Presented papers and conducted dialog at conferences

Table 6. Summary of Accomplishments

a. Amadeus (a metrics repository and analysis tool) is a STARS-supported tool from Amadeus Research; PSE (Process Support Environment) consists of two STARS-sponsored tools, SPMS from ccPE and ProjectCatalyst from SET (ccPE and SET are Loral/FS subcontractors), SoDA (a documentation generation tool) is from Rational, and ProDAT (an IDEF-oriented process definition tool) is a GOTS tool being developed via funds from the ESIP program of Sacramento, ALC.

b. Embedded Computer Resource Support Improvement Program (ESIP) is managed by Sacramento/ALC.

Labor Metrics Summary

Figure 11 provides a summary view of the labor expenditures from 10/1/93 through 10/31/94 (13 months) divided into the project's three SEE work breakdown structure (WBS) categories. 59% of the labor was devoted to completing the development of the STARS-sponsored PSE and evolving it to factor in early usage experience. (This labor was primarily consumed by Loral and its subcontractors.) 19% was devoted to System Management, which includes the logistics associated with procuring and installing the SEE as well as the system administration support of the SEE. 22% was devoted to SEE Engineering, which includes the technical aspects of designing and integrating the SEE solution.

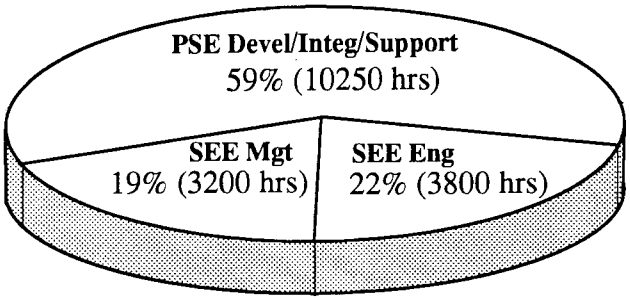


Figure 11. SEE Labor Summary Metrics (10/93 - 11/94)

Some introductory comments about these labor allocations are appropriate before moving into the more detailed discussion in the next subsection.

First, the PSE effort was a continuation of earlier work (including last year's Preparation Phase work) to bring the STARS PSE to fruition. This activity proved to be a larger effort than expected, largely due to the unprecedented usage paradigm targeted by the PSE. The experience related to the PSE is discussed in more detail in "Tool Development" on page 61, and "Process Support Tools and Integration" on page 65.

Second, the project had hoped to devote more attention to SEE Engineering but was constrained by the following:

(1) Unexpected levels of effort in the other two SEE WBS categories, notably the larger than expected PSE effort and the non-technical aspects of planning and procuring the SEE hardware, software, and networking, and

(2) Other project priorities, notably application engineering and process engineering, which constrained all three SEE budget categories.

4.4.3 Analysis

Acquiring Pre-Integrated Toolsets

The Demonstration Project benefited from selecting the Rational toolset (Apex, ROSE, SoDA), and the TRW toolset (UNAS, SALE, and the RICC tools[1]), since both toolsets were already on a path that matched the needs of the SWSC. The choice to use these vendors' solutions has worked out well.

Problems Encountered with Integration via BMS

We attempted to perform some control integration using the integration mechanism within IBM AIX SDR Workbench - the Broadcast Message Server (BMS). We quickly discovered that the BMS is intended for dialog communications, so relatively sophisticated programming is needed.

Development and Integration of Process Support Environment

At the beginning of the Performance Phase, the development of the Process Support Environment was behind schedule; but because of the perceived long-range value of the PSE, a joint decision was made to continue. Development continued until March, 1994. An extended pilot was conducted, and based on the pilot, usage began in May. Various problems were encountered, from bugs to immature interfaces, and a major improvement effort was launched in July. Both SPMS and ProjectCatalyst were upgraded to provide better integration: the ProjectCatalyst view of the process can be instantiated from the SPMS view. The testing of this second generation of the tool set has been completed, and the new versions of the tools are in pilot usage. The project has made a sizable investment in this toolset, both in the form of development

1. The TRW Reusable Integration Command Center (RICC) tools are Display Builder and RHMI (COTS), Query Builder and Query Processor (GOTS), and Message Builder and Generic Message Parser (GOTS).

effort and also in the overhead imposed on the application engineering activities as they worked through the problems and provided usage feedback.

Based on SEE survey feedback results, task leads have taken to ProjectCatalyst's concise spreadsheet-like way of viewing their delegated work in progress and appreciate the automatic reporting of status based on the engineering work starts and stops. Engineers have indicated that they appreciate the structured way their tasks are maintained in Process Weaver agendas, as well as the ability to quickly navigate to their work product files from their work contexts. Unfortunately, despite an apparently good direction for these tools, the initial bad experiences with immature interfaces and bugs has eroded confidence. Usage feedback has led to a significant overhaul of the PSE, and it is hoped that the second generation capability will provide much better support.

Connectivity among Team Members

The requirement for electronic connectivity was cited in the Preparation Phase section of this report, but progress toward resolution has been slow. The project has coped well with the limitations - e.g., software development productivity has exceeded expectations for the SCAI Release 1 effort - but it is clear that overall synergy could have been better if all project personnel had been linked on a common SEE from the outset. Connectivity with the Kaman Sciences group was achieved during this period by physically relocating the personnel. Connectivity with the TRW group was achieved by implementing a secure T1 link to the TRW facility.

Connectivity with Outside Organizations

The requirement for electronic connectivity with "the outside world" was cited in the Preparation Phase section of this report, but primarily because of security restrictions, little progress has been made. As of this report, the only connectivity with the outside world is via separate unclassified PCs, either via dial-up modems (9600 BPS) or via the base's SPACELAN network (effectively limited to 9600 BPS).

Pilots

The Demonstration Project conducted pilots of Amadeus, IBM CMVC, the STARS Process Support Environment (PSE) toolset, and ProDAT. Based on pilot usage, all of these were selected for use. Each pilot activity took quite a bit of energy - notably the PSE - but it is too early to tell if the value to the organization justifies the investment.

Tracking System

Although the Demonstration Project was late in establishing a common tracking system (seven months into the Performance Phase), it has now successfully adopted IBM CMVC. As of this report, however, only certain project areas have cutover to it: SEE system management, for work orders, and problem/enhancement tracking for managing the evolution of several SEE tools. For these areas, the process is working very well. Other project areas are still using largely manual means of tracking. In retrospect, the project would have profited by earlier concerted attention to defining and planning for this requirement, by enumerating the types of information (problems, action items, issues, experiences, etc.) that needed to be tracked, defining the recording and tracking process, and selecting a project-wide tool to support the process.

The automated tracking system is now supporting the following functions:

(1) System management work order processing

Prior to May 1994, all SEE change requests and SEE problem reports were in the form of System Management Work Orders, a manual paper system. These actions could range from moving a terminal from one room to another, to addressing a critical problem with one of the SEE's software products.

The use of IBM CMVC for SEE problem/change management was a significant improvement over the manual paper system. Once a problem/request was entered into the CMVC database, the system management team was automatically notified

via E-mail. The system managers then used CMVC to manage the problem/request resolution.

(2) SEE product problem/enhancement tracking

Certain products being used on the SEE are still in their R&D phase, such as the STARS PSE. For these products in particular, IBM CMVC provides an essential communication and tracking vehicle in working with the product developers to correct deficiencies and address usability improvement requests.

(3) COTS vendor feedback (partially implemented process)

IBM CMVC can also be used as a "suggestion" box for COTS vendors. Users can add a CMVC log entry to report desired product changes, and system management can forward consolidated reports to vendors for their consideration. This forum can also be useful for tracking key problems with COTS products. This particular use of CMVC requires a system management process change that has not yet been made.

SEE Surveys

Two SEE surveys have been conducted to date. See Appendix F, SEE User Surveying Experience, for an in-depth discussion of survey techniques and results.

4.4.4 Lessons Learned

As can be seen by comparing the objectives and plans for this phase with the actual achievements, the project experienced ups and downs in the SEE area - with some major achievements as well as some short-falls. This subsection briefly discusses and interprets relevant experiences, focusing on topics of particular interest to projects contemplating transitioning to megaprogramming. It should be noted that the project has prepared a more extensive Lesson Learned section, including additional lessons that are applicable to a wider audience - whether interested in megaprogramming or not. Space has prevented their inclusion in this report, but they are available upon request from Richard Randall, Loral/FS-G at (719) 554-6597.

The lessons are grouped into the following lifecycle activities:

- **Domain Engineering** - developing a strategy for the family of SEEs that will support a product-line organization.
- **SEE Management** - managing the administrative and engineering activities for a SEE, with emphasis on aspects present when providing a SEE for a significantly new way of doing business - such as megaprogramming.
- **SEE Architecture** - engineering a common architecture for a SEE product-line.
- **Platform and Tool Selection** - selecting hardware, networking and software tools for a new SEE.
- **SEE Integration** - combining the functions of the SEE so that there is reduced end-user perception of the SEE's individual components.
- **SEE Improvement** - gathering information about the SEE's functions, performance, maintainability, etc. and pursuing needed improvements.
- **Technology Transition** - transitioning new SEE approaches and techniques to SEE engineers and end-users.

Domain Engineering Lessons

This subsection covers lessons learned about developing a strategy for the family of SEEs that will support a product-line organization. In a megaprogramming organization, the SEE is viewed as a common asset, geared to support the organization's common process. Although the SCAI is the first application in the

future SWSC product-line, the Demonstration Project charter calls for laying the groundwork for the future product-line requirements. (Note that since the organization is still engaged in building the first system in the product-line, the lessons in this group are more aptly termed "observations based on experience but not yet proven.")

The reader is also urged to refer to Appendix G, "Megaprogramming and SEE Integration", which contains a more in-depth discussion of these concepts.

Lesson: *The development of an architecture for a product-line SEE requires a product-line mentality for the SEE itself - complete with SEE domain engineering.*

This objective - to support the future SWSC product-line requirements - has led to SEE thought processes which are analogous to the domain engineering thinking at the SCAI application level. Upon reflection, this is entirely appropriate: in the future application product-line organization, there will be multiple *logical* SEEs - one for each application being developed. These SEEs will share many common hardware and software assets, but each project will have a unique view of these assets which will be their own tailored realization of the SEE.

The requirements on these SEEs are dictated by the common process (the same process framework, but tailored realizations) and the common artifact construction methodology (same architectural infrastructure approach, but some tailoring in each application instantiation). Thus, to deliver end-user functionality "cheaper, better, faster" to the product-line organization, SEE engineers must perform true domain engineering of the SEE domain.

Looking back over the experience to-date with the SCAI SEE (and its forerunners prior to the Demonstration Project), it is clear that domain engineering thought processes have influenced the acquisitions and integrations that have led to our current SEE. Table 7 depicts some of our key SEE characteristics, cast in the light of SEE domain engineering. In each case, there is a clear parallel to the domain engineering results that have influenced the SCAI application's architecture, as discussed in Section 2.0, DE/AE.

Major Architectural Characteristics	SCAI SEE Domain Example Sub-Categories	SCAI SEE Domain Implementation Examples
Layered architecture	Common underlying operating system environment	Unix, TCP/IP, NFS, X
	Process support environment	STARS PSE
	Major functionality encapsulations	Rational's integrated family of Ada support tools (centered around Apex), TRW's integrated family of code generation tools supporting the architectural infrastructure (UNAS, RICC)
Common user interface	Common window-handling	Motif
	Common window behavior look and feel	Open Interface (from Neuron Data), Display Builder (from Rational)
Common program interface mechanisms	Low-level API services	Broadcast Message System (part of HP's Softbench)
	High-level data repository interfaces	COTS DBMSs (Oracle, Sybase)
Component reuse	COTS tools	Various

Table 7. Architecture Characteristics for the SEE Domain

Lesson: *(Hypothesis, strongly suggested by experience to date.) A product-line organization should establish a focal point SEE organization for the entire SEE product-line.*

Having an organization responsible for the SEE product-line ensures that the SEEs are managed as a product-line and that product line activities (such as domain analysis) are done. The SWSC SEPG established a tools "Integrated Process Team" (IPT) to address the proliferation of SEE tools across the SWSC directorates, and in doing so has acknowledged the problem of stovepipes within the SEE domain. As the SWSC

moves toward a product-line way of doing business for its end-user applications, it will need to move to a SEE product-line (discussed above) to support it.

SEE Management Lessons

This subsection contains lessons learned about managing the administrative and engineering activities for a SEE, with emphasis on aspects present when providing a SEE for a significantly new way of doing business - such as megaprogramming.

Lesson: *Managing change in the domain of the SEE is a key challenge for organizations transitioning to megaprogramming.*

For most organizations, transitioning to megaprogramming represents significant change - with corresponding pressure on the SEE to change to support the new ways of doing business. Well-meaning visionaries will seek the latest tools and technologies - and rightly so, since there are many gaps in proven tooling for megaprogramming (support for domain engineering, integrated process definition and execution, architecture-directed program generation, etc.).

However, such changes bring management headaches as well - such as the following, all of which have been encountered on the Demonstration Project:

(1) Planning uncertainty - due to such factors as slipped delivery dates and integration mismatches,

(2) SEE instability - due to the newness of the tools, including the possibility that a tool must ultimately be thrown out, and

(3) SEE technology transition issues - due to such factors as paradigm mismatches, extensive need to train users, and the possible need to adapt current processes to take advantage of new tool capabilities.

There is no formula for judging where to draw the line, but here are some weapons in the manager's arsenal:

(1) A SEE engineering staff that includes someone with extensive experience, particularly in the pitfalls of new tools,

(2) Piloting (discussed below), and

(3) Common sense.

The main point is to be aware that change is itself a key SEE challenge.

Lesson: *SEE Engineering "bandwidth" will be impacted by a surprising amount of non-technical work.*

An organization transitioning to a significantly new way of doing business (such as megaprogramming) must address a challenging mix of technical and non-technical SEE issues, with a surprisingly large percentage of this effort going to non-technical issues. Further, because the non-technical issues constrain the engineering solution, proportionately less engineering bandwidth is available to address the technical issues.

In our case, SEE engineering was complicated by such non-technical factors as:

(1) The degree of change (cited in the prior lesson)

(2) The continuing evolution of the processes to be supported by the SEE

(3) Significant changes in budget and timing

(4) The geographical separation of personnel. Initially, three major contractors were located at three different locations - although this is becoming less of a problem, since the team is being consolidated in one location.

(5) The classified nature of the SCAI application. This necessitated the SEE to be classified - which in turn degraded external electronic communication and diminished the ability of off-site personnel to support the project.

Lesson: *A project should place early priority on intra- and inter-project communication and coordination.*

Megaprogramming projects can expect to make significant technical strides in short periods of time - but only if communication and coordination processes are well supported by their common SEE.

Lesson: *Stringent security requirements represent a significant hurdle for an organization attempting to transition to a product-line way of doing business. If possible, the organization should seek to engineer its common domain architecture, its process, and its supporting SEE in such a way that most work can be performed on an unclassified environment.*

We have experienced a degraded ability to exchange information because our on-desk SEE assets cannot be connected to the outside world. Here are some of the megaprogramming-specific impacts:

(1) Overhauling technological approaches can increase the need for consultation and support from outside the organization. Such interfacing is often most efficiently done via electronic communication and data transfer that is readily available from practitioners' desks.

(2) A product-line organization may be developing applications of differing classifications - which may prevent the respective teams from sharing common SEE assets and communication paths. This, in turn, may thwart TT and/or sharing of assets within the product-line organization.

(3) To effectively address the stated goal of transitioning the organization's technology to outside organizations, the ability to conveniently exchange engineering data with the outside world is desirable.

Lesson: *A SEE "czar" is required for effective management of the SEE activity.*

In the face of the inevitable uncertainty in requirements, and the need for long lead times to accomplish acquisition and meaningful integration, appointing a qualified SEE "czar" early in the program is necessary. This lead person will be a single point of responsibility for the project's SEE and will coordinate both SEE engineering and planning, making it easier to force certain strategic decisions early - despite the inevitable lack of complete information.

Some key SEE issues that should be addressed early are

(1) Artifact definition and management, especially configuration management;

(2) Project-wide tracking support (problems, issues, action items, etc.); and

(3) Metrics instrumentation strategy.

It is also essential to have an experienced lead engineer devoted to such engineering functions as SEE architecture, tool selection guidance, integration, etc. Due to the criticality of the SEE to achievement of megaprogramming objectives, management should attempt to insulate this engineer from the host of non-technical problems that are sure to occur (discussed above). The conventional approach is to treat the SEE as an acquisition and maintenance problem; thus, the engineering aspects of the SEE tend to lag.

SEE Architecture Lessons

This subsection contains lessons learned about engineering a common architecture for a SEE product-line.

Lesson: *The architecture for a SEE to support megaprogramming must be developed incrementally.*

An outside consultant cannot present an organization with a pre-integrated SEE to satisfy product-line objectives. There are too many variations; too much needs to be customized.

At the start of the Preparation Phase (10/92), there was an initial perception that Loral, on behalf of STARS, would be providing an off-the-shelf SEE that would be fully integrated by the start of the Performance Phase. In retrospect, this was naive.

The design of the SEE is predicated on a clear statement of the requirements, namely the automation requirements to support the application product-line. These requirements, in turn, are dependent on the organization's understanding of its process. In the case of the Demonstration Project, some aspects of the

basic approach were still being worked out at the beginning of the Performance Phase - let alone the process.

Thus, by the end of the Preparation Phase (at the time of the first Experience Report), Loral and the Air Force were able to cite one of the major lessons learned thus far on the Demonstration Project so far - one that transcends the SEE: **All aspects of the product-line approach - domain models, process, application architecture, and in particular the SEE - must be built-up incrementally.** An incremental, iterative approach is necessary since the organization will be simultaneously grappling with fundamental approach issues - delaying the availability of firm SEE requirements.

This principle is recognized by [Brown92], who argues for "evolutionary approaches, rather than revolutionary."

Platform and Tool Selection Lessons

Lesson: *Heterogeneous open-systems platforms provide flexibility that can at least partially offset the additional maintenance overhead.*

The SCAI SEE includes both Sun and IBM RISC System/6000 platforms. The Suns were part of the existing SEE before the Demonstration Project, but the project decided to complete the SEE with IBM platforms and to network the Suns and IBMs together. Although the two platforms cannot run each others' binaries, it is commonplace to network their file systems together and to remotely execute applications via XWindows/Motif.

Although having the two platform types in the SEE meant that maintenance and system management was more complicated, the project has realized some important benefits. New tools and new versions of tools are often released on one platform (usually the Sun) and then ported to others - and there can be quite a bit of time between ports. The project has thus been able to take early advantage of new releases of several tools during this period, including Framemaker, Rational ROSE, and Rational SoDA. It has also been able to take advantage of tools that run only on one platform, such as the STARS PSE (IBM platform), and ProDAT (Sun).

Thus, the additional flexibility provided by heterogeneous platforms can at least partially offset the additional maintenance overhead.

Lesson: *Pilots are an important vehicle for both tool selection and tool usage.*

If there is no past experience with a tool, the best method of in-depth evaluation is to conduct a pilot. A pilot is a precursor activity to try out a new tool to determine:

(1) If its functionality and usability will effectively satisfy project objectives;

(2) How best to use the tool in the context of the organization's process and its other tools; and

(3) How to best integrate it into the SEE.

The pilot should be based on realistic usage scenarios. The best pilots are essentially precursor production usages on non-critical path activities.

On the SCAI project, we piloted Amadeus, IBM CMVC, the STARS Process Support Environment (PSE) toolset, and ProDAT.

Lesson: *A project must be judicious about the number of new tools.*

Organizations pursuing a significant new paradigm (such as megaprogramming) will have a desire to use new tools - sometimes advanced tools whose practicality has not been widely established. These organizations face the challenge of balancing the legitimate need for such tools against the significant organizational energy required to first evaluate them and then, if selected, to assimilate them into actual use. Two important factors to take into account are the energy to modify existing methods and the risk of usage problems (e.g., bugs).

Lesson: *A project should be cautious about committing to tools still under development.*

Most of the tools selected for use on the Demonstration Project were COTS. Exceptions were the process support tools comprising the STARS PSE. The PSE is the culmination of several years of process automation R&D, and its development continued throughout the first year of the Performance Phase.

SEE Integration Lessons

Lesson: *High payoff integration can result from selecting a vendor whose toolset evolution objectives align with your organization's goals.*

Two examples of how this strategy has paid off for the Demonstration Project are the Rational toolset (Apex, ROSE, SoDA), and the TRW toolset (UNAS, SALE, and the RICC tools). In both cases, the Air Force had already established confidence in the vendors, and their toolsets were already on a path that matched the needs of the SWSC product-line objectives. In the case of Rational, the SWSC is committed to Ada (supported by Apex), OO modeling (supported by ROSE) and documentation automation (supported by SoDA). In the case of TRW, the SWSC is committed to the underlying architectural infrastructure supported by the TRW toolset. The Air Force choice to use these vendors' solutions has worked out well, since both have since pursued a vigorous product improvement cycle, including progressively higher degrees of integration.

Lesson: *Acquiring a single vendor toolset is usually more cost-effective than integrating separate tools.*

The foregoing discussion of the Rational and TRW toolsets illustrates that off-the-shelf integration has worked quite well for this organization. Although there has been relatively little home-grown integration to date, experience suggests that the following advantages and drawbacks should be weighed:

(1) Advantages of off-the-shelf integration:

- Your organization will require less maintenance labor/expertise in-house.
- The fewer vendors to deal with, the less logistics hassles; the fewer vendors, the less glue. Integration complexity increases as the number of tools/vendors increases.
- The vendor takes care of assuring the toolset remains integrated as components change; whereas if the integration is the responsibility of the local organization, there are considerable headaches when one of the integrated components changes.
- With more concentrated investment in a single vendor's product (as opposed to scattered investment in numerous vendors), you have more leverage in getting the single vendor to accommodate your specific requirements.

(2) Drawbacks of off-the-shelf integration:

- You have to accept the vendor's integration, and you still have to solve the integration problems with other tools.
- All of the component tools in the integrated toolset have to be updated at once.
- Sometimes the vendor's plans do not match well with your own schedule. For example, if an underlying OS or DBMS has to change, you will have to wait, as we had to, for the vendor's total package update, instead of a single component's update. If you are doing your own integration, you can swap out a piece (reasons for swapping: licensing issues, functionality, etc.).
- It may be hard - or even impossible - to integrate a component of a monolithic integrated toolset with an outside tool. A monolithic toolset may have missing functionality that must be provided by another toolset that has overlapping functionality with the first - resulting in wasted redundant functionality in the SEE. (An example of this currently is Rational Apex, which supports version control but not problem tracking; IBM CMVC, which also supports version control and problem tracking but does not provide Apex's semantic understanding of Ada.)

Lesson: *Maintaining process information in multiple forms poses a significant maintenance challenge.*

On the SCAI project, process modeling and enactment is split among three tools. ProDAT supports activity-based modeling, using IDEF notation as a basis; SPMS supports workflow modeling and process driven planning, using ETVX notation as a basis; and ProjectCatalyst supports workflow execution and coordination, using task lists and agendas as a basis. Each tool provides its own independent method for entering and storing its data, and each supports a different set of process modeling notions. Using all three tools allows the project to take advantage of the unique capabilities of each but has the drawback that process information is split and must be managed carefully to keep it synchronized.

The SCAI is grappling with this problem now, and some remedial action is in progress. First, SPMS and ProjectCatalyst were already partially integrated and share a common relational database management system for their data. In addition, both have recently been upgraded to provide better integration: the ProjectCatalyst view of the process can be instantiated from the SPMS view. Second the potentially voluminous combination of IDEF diagrams and supporting text is being consolidated into a single database via ProDAT. Hopefully, these two measures will greatly reduce the maintenance of SCAI process information, which will in turn increase the likelihood that the processes will actually be executed in accordance with their definitions - thereby enabling a genuine process improvement cycle.

Lesson: *Artifact management (notably configuration management) is a central integration issue.*

Since the SEE's primary role is to provide the means to create, maintain and access project artifacts, configuration management is a pre-eminent SEE integration issue. At the time of this report, CM support on the SEE is still fragmented. For example, IBM CMVC is in use for many aspects of project work (e.g., most problem tracking), and Rational Apex CMVC is in use for Ada development (code management only, no problem tracking), but as of yet, there is no SEE integration between the two - although some prototyping has been conducted to pave the way for future integration.

There are several instructive reasons why integration has been lagging in this key area:

(1) The CM process is only partially defined. There is a high-level $IDEF_0$ description, but the project is still grappling with nuts-and-bolts definition of artifacts, how they will be constructed and managed, how they will be delivered from point to point within the process, and how the directories should be set up on the SEE to support this work. (A complicating factor is that developing the SCAI's new megaprogramming approach has involved coming to grips with new types of artifacts, combined in new ways.)

(2) A number of candidate CM tools must be considered, three of which are already in use on the SCAI (IBM CMVC, Rational CMVC, the State Data Repository built into ProjectCatalyst), and many more currently in use elsewhere within the SWSC.

(3) CM-implications of defining a product-line process are not well understood.

(4) Available SEE expertise has been addressing other priorities.

Since the project's work is already well underway, the incomplete state of the CM architecture for the SEE poses a problem. Each activity has developed a local CM process that is effective for their work, but as upcoming deliveries are made for software increments, and as parallel engineering takes place on successive versions of artifacts, pressure is mounting to develop an integrated CM solution.

Lesson: *Use of Broadcast Message Server (BMS) messaging services required unanticipated engineering effort to provide guaranteed message delivery.*

The integration mechanism within IBM AIX SDR Workbench - the BMS - is intended for dialog communications, so relatively sophisticated programming is needed on both sides. The way we programmed the use of Workbench messaging service did not force the start-up of the of the receiver of the message. As a consequence, some messages were undelivered, and integrity of the integration between the tools was lost. We now understand the requirement for more handshaking and error checking within the message event loops.

There were other limitations in Workbench that surfaced during the SEE integration effort. Communications between tools were supported only within a single user session. The IBM Workbench product is an

implementation of the Hewlett-Packard BMS technology. We believe many of the Workbench limitations were the result of the lagging implementation of BMS improvements.

The emerging standard for SEE integration messaging is found in Sun's ToolTalk [ToolTalk94]. This technology is part of the emerging industry standard referred to as the Common Desktop Environment (CDE). All the major environment platform vendors (Sun, Hewlett-Packard, IBM, Digital, and others) have agreed to provide CDE compliant products with their platforms. We believe that future releases of AIX will provide an implementation of BMS (through ToolTalk) that will ease most of the limitations that we encountered.

Lesson: *A Process Support Environment can provide an effective SEE integration layer.*

[Brown92, p. 304] discusses the fundamental role an organization's process plays in establishing requirements for a SEE: "Understanding the process must precede introduction of solutions." Prior to its affiliation with the Demonstration Project, Loral conducted extensive research and development in the area of process automation. This work lead to the notion of a "process support environment" (PSE).

The PSE is a set of tools which supports the organization in defining its process and then using the process as the basis for SEE-assisted planning and execution. Experience in developing and using PSEs - including the recent experience on the Demonstration Project - has led to the view that it is useful to think of the PSE as an abstract integration layer (see Figure 12), which encapsulates the organization's process and exports the SEE's functionality to the end-user in the context of the process. This layer is only partially realized to date, but experience has already shown its potential for implementing process-driven SEE integration.

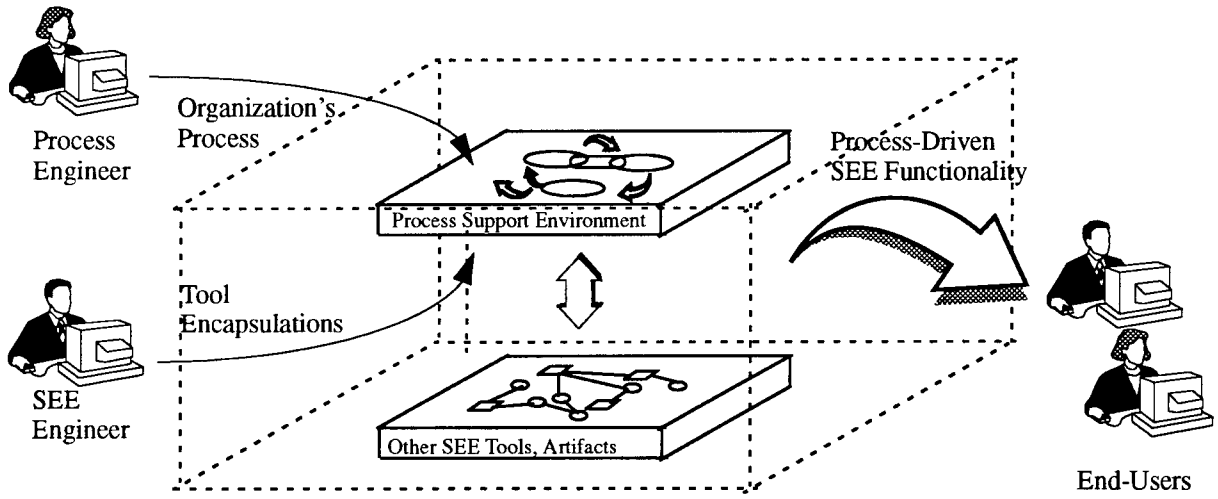


Figure 12. PSE Viewed as a SEE Integration Layer

The PSE layer in the diagram includes SPMS, ProjectCatalyst, and Process Weaver. SPMS is used to model the process using a graphical ETVX[1] notation and to construct a process-driven plan for the project. ProjectCatalyst then imports this plan and presents it to managers and task leads as a hierarchy of process components that are used to manage the tasks called for by the process. As each task is dispatched to the designated engineer, ProjectCatalyst supplies a "work context" to Process Weaver, providing details of the artifacts to be accessed and the SEE tools to be used to work with them. To enable Process Weaver to launch the tools against the artifacts, the SEE engineer provides small tool encapsulation scripts.

Thus, the process engineer instantiates the PSE with the process, and the SEE engineer instantiates the PSE with the low-level tool integrations.

Interested readers are urged to refer to Appendix H, Using Process to Integrate SEEs, which is a technical paper that provides an abstract model for a PSE and maps the SCAI PSE to the model.

1. ETVX - Entry, Task, Validation, Exit - a notation for depicting process sequencing.

SEE Improvement Lessons

This subsection contains lessons learned in determining how well the SEE supports the users, as well as implementing appropriate improvements.

Lesson: *Medium to large software development projects (team of 10 or more), need a SEE-supported problem tracking system.*

Lesson: *Management of the SEE can be enhanced via an automated tracking tool.*

Lesson: *A SEE improvement process should include provisions for organized collection of feedback to COTS vendors on how to improve their products.*

An automated tracking system should track not only problems but also action items, issues, and experiences. This system should be well-defined and implemented early in the project. Our project has adopted IBM CMVC for this type of support, but as of this writing, the project has not completely transitioned to its use. For example, SCAI application engineering is not using it to track problems found during walkthroughs and integration testing.

Lesson: *Users find the native interface of IBM CMVC cumbersome for routine use.*

A key strength of IBM CMVC is that it enforces a fairly complete process in which problem tracking is tightly integrated with artifact changes. However, the drawback is that the process seems unnatural and unduly cumbersome to many users. CMVC provides a fair amount of tailoring capability, and some work has been done to adapt it to the SCAI users. In addition, several training sessions have been held to acquaint users with the process and the tool's use. Finally, work is in progress to add a user-friendly front-end to CMVC that focuses on the most frequently encountered process aspects.

These user perception problems have been partially responsible for the acceptance problems cited in the prior lesson.

Lesson: *Process-driven databases are a good basis for metrics.*

Process-driven tools can provide a credible source of measurements to support metrics, since the measurements are driven by the work itself.

On the Demonstration Project, there are two notable examples where this principle is being applied:

(1) Process-driven planning and execution.

ProjectCatalyst tasks are tied to the SPMS-generated plans - which in turn are based on the project's defined process. As tasks are executed by practitioners, the start and completion dates are automatically posted to both the ProjectCatalyst and the SPMS databases. This means that task leads and managers are provided accurate visibility of work progress - since milestones are reached only in accordance with the defined process, complete with prescribed validations.

In addition, as tasks are completed, the SEE gathers task wall-clock time automatically and expended labor statistics via prompts to the user. These are also posted to the ProjectCatalyst and SPMS databases and are available for metrics analysis.

In both cases, the accuracy, timeliness and completeness of the data is greatly improved by the fact that it is directly tied to the work itself.

(2) Problem and change tracking.

The project is using an automated tracking system to track not only problems but also suggested improvements and work orders - which are entered into the system as they are conceived. The system tracks their progress as problems are resolved and work orders implemented. We can quickly compute metrics on the progress and easily identify problem areas.

Here again people don't have to do anything special: the accuracy, timeliness and completeness of the measurement data is derived from the fact that it is tied directly to work processes.

Lesson: *Periodic SEE end-user surveys are an essential part of an organization's SEE improvement process.*

Lesson: *SEE end-user surveys should request feedback on individual tools, SEE integration among the tools, and system management support responsiveness. The requested feedback on tools and integration should cover both current usability and utility as well as future direction.*

Active process improvement and SEE improvement programs are central to megaprogramming. To identify needed improvements, the SEE's end-users must be regarded as customers. We have conducted two SEE surveys to date. Feedback was sought on individual tools, SEE integration among the tools, and system management support responsiveness. Please refer to Appendix F, SEE User Surveying Experience, for an in-depth discussion of survey techniques and results.

The following are some of the survey findings:

(1) Users felt the SEE's functionality was adequate for their work and "headed in the right direction."

(2) The areas of ***tool functionality*** that were deemed "strategic" to the project's long term objectives were:

- Ada software production (currently supported by Rational Apex and related tools),
- Architectural infrastructure support (currently addressed by the RICC tools),
- Object-oriented and information modeling (currently addressed by ROSE and Teamwork/IM),
- Metrics,
- Process definition (currently addressed by ProDAT and SPMS), and
- Documentation and automated documentation production (currently addressed by FrameMaker and SoDA).

(3) The areas of ***SEE integration*** that were deemed "strategic" were:

- Configuration management - with most other SEE areas,
- Project management - with most other SEE areas,
- Metrics - with most other SEE areas,
- Object-oriented and information modeling - with document production and Ada software production,
- Ada software production - with architectural infrastructure tools and software certification, and
- Process modeling - with project management and process enactment.

Based on the analysis of the survey results, the following survey improvements will be taken into account for the next survey:

(1) A larger cross-section of the end-user population will be surveyed (40% versus 25%).

(2) Additional judgements will be requested to enhance the survey's utility in deriving metrics. For example, people were asked to assess the importance of functionality groupings to their own jobs; they should also be asked to estimate the importance to the future product-line objectives.

(3) Users will be asked to rate the value of the survey itself - as well as to provide suggestions for improvement.

(4) Outside consulting will be sought to review survey techniques and recommend improvements.

Technology Transition Lessons

Lesson: An organization seeking to transition a significantly new SEE approach into practice must use a variety of technology transition (TT) avenues.

When planning SEE activities, the importance of technology transition (TT) should not be underestimated - to transition the new functionality to the project, to transition the overall SEE approach to the parent organization, and to communicate the approach to outside observers. Unfortunately, labor metrics during this period did not provide sufficient granularity to provide quantification of the TT portion of the SEE effort, but the following list of SEE-related TT activities suggests their potential scope - as well as some specific channels that can be used:

(1) Classes and Pilots (such as training for the PSE, IBM CMVC, SoDA, etc.).

(2) Presentations and demonstrations within the SWSC (such as a briefing to another SWSC directorate about the project's use of IBM CMVC).

(3) Active involvement in SWSC-wide Software Engineering Process Group (SEPG), allowing Demonstration Project experience to contribute to the long-range thinking of the SWSC as a whole.

(4) Demonstrations of the SEE to outside organizations in forums such as:

- Software Technology Conference 1994, Salt Lake City, Utah (coordinated demonstration booths were STARS, SCAI Demonstration Project, and Sacramento/ ALC), and
- STARS Demo Days, STARS Technology Center, Arlington, Virginia.

(5) Presentations and discussions with other STARS Demonstration Projects.

(6) Outside Conference Representation, such as:

- Delivered presentation and participated in full day discussion on SEE integration experience at the NIST Open Systems Environments Implementors' Workshop (OIW); the material was based heavily on SCAI experience, and
- Prepared two SEE papers to be presented at Software Technology Conference (STC '95), and one to be presented at the 1995 SEPG conference.

4.4.5 Summary

According to our recent SEE survey, the Demonstration Project SEE is serving the end-users reasonably well and appears to be headed in the right direction. The main state-of-the-art technology thrust of the Demonstration Project is process and SEE support for process. The project has faced significant challenges in both, and it is clear that the incremental, iterative approach is the only way to go. There are promising results to date, and much interesting work left ahead as we work towards a SEE to support a full SWSC product-line.

Judging from our experience to date, the highest payoff path to effective SEE integration is to adopt strategic toolsets from vendors whose long-range product vision is aligned with the SWSC's product-line objectives. In our case, these vendors are Rational (Ada, OO modeling, automated documentation production) and TRW (architectural infrastructure support). Even with such toolsets, however, the product-line organization must have its own SEE engineering expertise to customize and tailor off-the-shelf tools, to implement integration at the total SEE level, and to plan and implement continual SEE improvement as the product-line needs evolve.

5.0 Metrics

5.1 Introduction

Metrics are instrumental in providing information to quantify software life cycle cost, quality, and capability differences resulting from the presence and absence of megaprogramming technologies. The SCAI project will quantitatively and qualitatively measure the effects of megaprogramming on the development effort and the resulting product. During the Preparation Phase, the SCAI team formulated a preliminary metrics process. The foundation for the metrics process is the formally defined set of project goals. The team defined the project goals (refer to Section 1.1.3, Summary of Demonstration Project Objectives) and began detailing the metrics collection and analysis approach.

5.2 Long Term Objectives

The goal of the SCAI Project metrics activity is to establish a measurement process along with the necessary organizational and technological support to:

(1) Enable the evaluation of progress toward achieving the goals of the SCAI project,

(2) Support continuous improvement of the processes being applied, and

(3) Provide a historical artifact to be used by future projects in determining the potential benefits of applying the SCAI megaprogramming approach.

5.3 Preparation Phase

5.3.1 Plans

Preparation Phase Objectives

During the Preparation Phase the STARS project metric objectives were to:

(1) Define metrics process (approach).

(2) Gain buy-in on SCAI goals from project personnel.

(3) Prepare for long term objective one above:

- Document initial set of measurements.
- Begin capturing some data.

Assumptions

The following assumptions were made at the start of the metrics effort:

(1) Initial guidelines and methods would come from:

- DoD Core Measurements.
- IDA 12/92 prototype Measurement Plan.
- Goal/Question/Metrics (GQM) Method.

(2) Some staffing would be available during the Preparation Phase (1 Full Time Equivalent person of effort from a 3 person team).

Planned Activities

The Metrics Team decided early in the Preparation Phase to combine the metrics work with the project's overall goal-setting work, with the rationale that the primary basis of any measurement activity is to help

determine how well the goals are being achieved. During the Preparation Phase, the team evolved the following process description to depict the integrated goals and metrics activities.

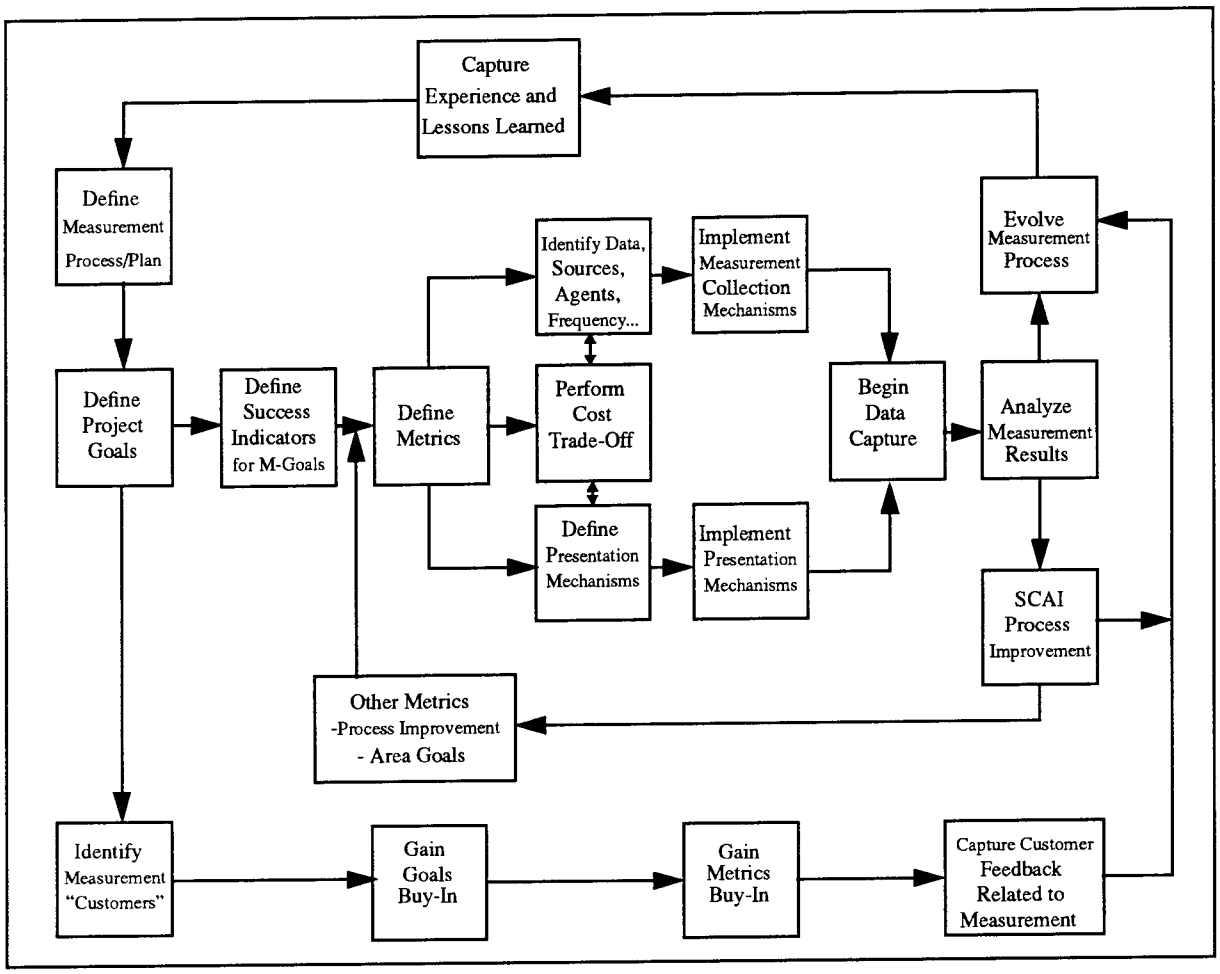


Figure 13. Metrics Process

The measurement process as shown in Figure 13 includes:

(1) Selecting SCAI Project top level goals.

(2) Identifying a set of success indicators associated with each goal.

(3) Defining a set of metrics which could verify the successful achievement of the goal.

(4) Establishing an approach for capture and collection of the metrics.

(5) Defining an approach to presenting and analyzing the measurement results.

(6) Instituting a continuous improvement cycle for the measurement process itself.

The approach to metrics definition is both top down and bottom up. From a top down perspective the project goals will be defined and refined. The purpose of the metrics activity is to determine if the goals are being reached. The goals provide the rationale for the actual measurements collected. You could also think of the goals as providing requirements on the metrics activity.

From a bottom up perspective, however, there is a need to determine what specific data can be gathered and how/where it will be captured. The guidance and templates provided by the DoD Core Measurements are used to help define these specific details.

A cost benefit trade-off will be performed to determine what measurements are practical. It is at this point that the top down and bottom up approach meet. The result will be an affordable metrics approach that provides a tool for project management, process improvement and future decision making regarding the application of megaprogramming.

Also shown in Figure 13 is the need to identify and gain buy-in from the "customers" for whom the metrics processes are being performed. These customers include project management from the SWSC, STARS and ESC as well as all the SCAI project participants.

A key element of all aspects of this metrics process is the consensus building approach associated with each of the activities. The process of defining goals and success measurement can be very effective in team building and flushing out the overall project process. To this end, each aspect of this approach will be developed with the involvement of all project personnel. Each iteration of the process will be developed using interactive group techniques to assure group acceptance.

5.3.2 Summary of Accomplishments

(1) The SCAI Metrics Plan was published in October, 1993.

(2) Process (approach) was outlined and initiated.

(3) Version 2 of the Goals was documented and reviewed by project.

(4) Version 1 of the success indicators for 2 of the 3 top level goals was documented and reviewed by project. Goal 3 (Technology Transition) was deferred until mid way through the project.

(5) The initial draft of desired measurements produced metrics covering about 50% of the needed measurements. Many specific details need to be defined prior to capture of these measurements. Much of this detail will eventually be part of the SCAI processes.

5.3.3 Analysis

For each of the five points noted here, the specific lessons learned are described in Section 5.3.4.

We went in search of the SCAI processes and couldn't find them documented, so we defined and documented what we know we want to measure, and we still need to define how we will gather measurements.

(1) We were able to refine Goals/Subgoals and Success Indicators by meeting in small focus groups of 2-5 interested people for 3 hour meetings.

(2) We have taken the GQM approach and adjusted it to construe the questions as success indicators.

(3) We established collection categories, which appears to make the metric derivation and implementation approach more manageable.

(4) We were able to start this process with a small team of people and a reasonable amount of effort (1 FTE), to get an initial understanding of measurements needed on the project.

5.3.4 Lessons Learned

Lesson: *The level of process definition and the precision of metric definition are interdependent.*

Ideally, we would have wanted to say "Make sure processes to be measured are well specified" prior to defining metrics the process and product metrics the project will capture. However, what we really think is possible is that an organization can:

(1) Let the metrics support the process definition activities. The need to collect key pieces of information often highlights critical points in the process definition which need attention.

(2) Include more human interaction when collecting information to be used to define measurements. Even if the process is not completely defined, humans can often predict what is needed from a measurement standpoint. Later process definition detail may allow refinement of the measurement requirements.

(3) At some point stop metric definition activities and put more effort towards process definition. There is a trade-off to be made here. If your metrics definition gets too far ahead of the process definition, the efforts may become redundant, resulting in an inefficient use of resources. It might be better to transition resources to the process definition activity and then pick up the metrics implementation when the process has stabilized.

Lesson: *Transitioning to a metrics-based approach requires early focus and ongoing reinforcement.*

Help the project team focus on what they intend to accomplish as early as possible, and have them identify how they would know if they have accomplished their intent. This early focus is important not only for overall project goals, but also for any goals associated with project phases. Once you have identified where the team wants to go, give them continuous visibility into where they are, using graphic displays.

Lesson: *People may view the new emphasis on metrics as a threat.*

The following questions were used with the team, during the Preparation Phase, as a vehicle to help clarify the role of metrics on the project and how metrics relate to individual team members' jobs:

(1) How do you relate to or feel about measurements?

(2) If you were sure that the results of measurement would not be used as a basis of retribution, what possible value might visible metrics contribute to your own job performance?

(3) What, if anything, is missing in the organization that would allow people to feel confident that measurement results will not be used against them?

Emphasize that the value of metrics to team members is the insight they provide into:

(1) the current level of performance.

(2) whether the current level of performance will allow the team to achieve its pre-specified objectives.

With visibility into what has been accomplished, given what one intends to accomplish, team members have the ability to take actions that can specifically address getting back on course when deviations occur. However, in order to be able to accurately report what has/has not been accomplished, the culture within the organization needs to be such that individuals and teams do not fear retribution.

In general, we found that people's attitudes about metrics were positive and supportive.

Lesson: *Goals/Success Indicators are critical and require consensus.*

It is never too early to define goals. Everyone on a project assumes they understand the goals. Unfortunately, put these people in a room and start discussing the goals and you find that there is a great deal of confusion and disagreement.

The process of articulating goals leads to team and consensus building. Much of the initial work done by the SCAI metrics team, related to defining goals and success indicators, should be carried out by a project regardless of any needs related to metrics collection. We used a technique of inviting small groups of project personnel to multiple half-day (about 3 to 3.5 hours) meetings that focused on a subset of the project goals. The subset of goals and the individuals invited to the meetings were matched by composing project groups along lines of the processes they performed, and then "mapping" groups to goals by determining if the performance of activities within their process was critical to meeting that project goal.

The approach worked well. There was sufficient time for detailed, and often frank, discussions to take place. We held two rounds of meetings (5 different groups in each round). The first round concentrated on goals and the second round refined the goals into success indicators. We captured issues from the first round in order to avoid getting too detailed and too hung up on definition. We found that by the second round most of the issues had resolved themselves and that the goals meetings were a catalyst to forcing discussions on important topics.

The risk we accepted with this approach was that not all project personnel were able to participate in the detailed discussions on all the goals. It is critical to do a careful job of matching people to goals for this approach to succeed.

Lesson: *Success indicators/questions are effective for refining goals and goal definition.*

The questions (from the GQM method) were worded such that a positive answer would provide an indication that we may be successfully achieving the goal. This gave the questions a very specific focus and helped to better clarify the goal.

Lesson: *The approach tends to explode, so the team needs to work constantly to aggregate.*

We first defined three top-level goals, then sub-goals and sub-sub-goals. The success indicators continued this expansion. It became clear that we needed to constantly try to aggregate the information and not let this expansion become unmanageable. We knew that the measurement definition would eventually pull things back together. But the concern was that the process of expansion would get out of hand, and you would never be able to trace all the goals and success indicators back to the specific measurements.

Lesson: *Collection categories made the process more manageable.*

We noted during the process that there would be four basic approaches to capturing the measurement data:

(1) Automatically via the SEE.

(2) Extracted from financial systems (staff hours/cost).

(3) Manual inspection of work products.

(4) Surveys of project personnel and potential SCAI users.

This made the organizing of the measurements much easier and allowed us to divide up the work of implementing the collection mechanisms.

Lesson: *A significant amount of effort is required for metrics definition and collection, and dependence on the project infrastructure and process definition is clearly evident.*

We were able to do the following activities with a group of three people spending approximately 1/3 of their time each on:

(1) Defining a metrics plan/process,

(2) Coordinating the refinement and consensus building for the goals and success indicators, and

(3) Developing a first draft list of the needed measurements and mapping them to the success indicators.

The effort was spread over a five month period, May through September (total of 5 person-months). However, at this point the workload increases significantly. The remaining activities are: defining specific measurements, data values, frequencies for collection, mechanisms for collection, database capture definitions, and presentation formats. The level of effort for this activity is estimated at 2.5 persons.

5.4 Performance Phase

5.4.1 Plans

Performance Phase Objectives

During the Performance Phase, the metrics team's objectives were to:

(1) Establish the SCAI metrics program.

(2) Develop unobtrusive collection mechanisms for metrics data.

(3) Provide meaningful metrics output, available on-line or displayed in every person's work-space.

(4) Develop a strategy to demonstrate Return on Investment (ROI) for the SCAI's approach to megaprogramming.

Planned Activities

To meet the objectives, the team planned the following activities:

(1) Continue the CIM[1] SWSC Software Maintenance Process Analysis by developing functional economic analyses (FEA) and generating appropriate business case documentation.

(2) Revise the Metrics Plan to accurately reflect the work being done.

(3) Refine the metrics process by applying SCAI process definition technologies.

(4) Identify SCAI process and product measurements and their data collection mechanisms.

(5) Adapt Amadeus, a metrics collection tool, to project needs.

(6) Collect the following project data:

- Labor-hour expenditures,
- Hardware and software defects,
- Tool usage statistics,
- Software size and complexity metrics, estimates and actuals, and
- Schedule metrics, estimates and actuals.

(7) Produce quarterly metrics reports providing metrics status and summarizing the collected data.

5.4.2 Summary of Accomplishments

This section details the accomplishments of the metrics team in executing the metrics plan. Both activities planned during the Preparation Phase and activities added during the Performance Phase are described here.

The following list parallels the list of activities in section 5.4.1 and describes both what has been accomplished and what has yet to be done.

(1) Completed the Activity Based Costing Foundation Workshop and preliminary FEA. More information is contained in a later section relating this to the computation of ROI.

(2) The metrics plan was not revised.

(3) Used SCAI process definition technologies to refine the metrics process.

(4) Reviewed SCAI processes and embedded definitions of process and product metrics.

(5) Implemented expended labor hours collection using Amadeus. Also developed the ability to collect tool usage metrics.

(6) Initiated collection of project data.

(7) Published three metrics reports. Reported effort data by work breakdown structure activity. Summarized problem tracking data relative to the SCAI SEE.

1. See Appendix B "Technologies Contributing to SCAI".

5.4.3 Analysis

Definition

In December of 1993, a process definition class was held in which project team members were taught what information was required to define enactable processes and how to use the SCAI process definition technologies to capture, record and access processes for project enactment. The metrics team defined the metrics process through the first level of process definition using $IDEF_0$ modeling techniques and the Process Definition Information Organizer Templates. Further formal definition of this process is being deferred due to resource constraints.

The team worked with the other four process definition teams (specification, incremental development, certification, and configuration management) to identify metrics that each process would collect and analyze for process and product improvements. These metrics were identified in a "best guess" manner, since process artifacts were not completely understood or defined given the newness of the process definition activity. Application and Domain engineers were also unfamiliar with performing the newly created SCAI DE/AE process.

As our processes mature, we are refining and adding to the key set of metrics. We have identified some metrics that each process could collect; however, except for effort data collection (and by time of publication of this report, tool usage statistics, connect time, walkthrough metrics, CMVC data on products in the SEE, and software complexity) we have not formally defined collection mechanisms.

In addition, the metrics team participated on the SEPG Metrics Integrated Process Team (IPT) to assist in the development and definition of SWSC-wide core metrics. The SWSC is the parent organization to SMX. Although the SCAI metrics goals differ from those of the SWSC in general, this interaction will eliminate the potential for duplication of effort when complying with SWSC metrics collection and reporting. It also provided a conduit for introducing the SWSC to some of the SCAI processes and technology.

Of the SWSC metrics, the SCAI team is currently collecting labor hours expended. The core metric categories that the SWSC is using have been evaluated to define possible additional metrics in each of the major SCAI megaprogramming functions. For example, the SWSC collects incident report (IR) status against a version of an operational system whereas the SCAI project is currently capturing incidents identified against products within the SCAI SEE. Only when the SCAI project becomes part of the SWSC operational software (after the end of the Demonstration Project) will IR status of the developed system software be measured.

Also, early in the Performance Phase, the STARS Demonstration Projects were encouraged to develop an approach to measuring the effectiveness of megaprogramming by computing the ROI for the STARS technologies. A strategy for computing ROI has become an added objective of the metrics team. Because of the difficulties in computing SWSC baseline productivity given the improvements made in the SCAI demonstration environment before the introduction of STARS technologies, a completely satisfactory approach to computing ROI has not yet been established. Our current approach to ROI computation is documented in the following section.

STARS/SCAI ROI Business Case Development

As the SCAI team identified the SCAI Domain/Application Engineering Process, the SWSC, under the auspices of the SMX Corporate Information Management (CIM) Business Process Improvement work group, continued definition and analysis of the 'AS-IS' SWSC Software Maintenance Processes. The team's objectives were to evaluate current SWSC Software Maintenance Processes and appropriate software development and maintenance alternatives from both a functional and economic perspective[1], developing a business case to support the preferred alternative.

Activity Based Costing was performed on $IDEF_0$ activity models of the software maintenance process currently used in the SWSC to establish the cost baseline of doing business today. This methodology applies costs to activities, it characterizes the value of each activity and ranks activities for improvement. This pro-

1. Corporate Information Management (CIM) Business Process Improvement Analysis of SWSC Software Maintenance. Activity Based Costing (ABC) Foundation Workshop 16 November 1993 to 10 January 1994.

cess of decomposing activities provides an approximation of the costs of current operations, including civilian and military labor, operations and maintenance, facilities, and materials.

A preliminary FEA was then performed to evaluate the costs of alternative ways to accomplish the functional objectives of the SWSC Software Maintenance Process. The two alternative approaches currently being considered are the STARS/SCAI approach and an approach that proposes outsourcing software maintenance and development to the maximum extent possible. Costs of doing business today and projected costs of doing business in future years have been developed for both the current SWSC software maintenance process and the STARS/SCAI alternative.

A significant cost factor in the current SWSC software maintenance process is attributed to the redundant development of code due to the lack of a single software architectural infrastructure. STARS/SCAI process predictions for improvements in costs, when the SWSC utilizes them, are currently based solely on the savings available through reuse of code given a single SWSC software infrastructure. They do not currently account for savings associated with improvements in process or software engineering environments. Over the next year actual data collected using the SWSC and STARS/SCAI metrics will validate the cost models and refine out year predictions. Models will be updated to reflect the costs and savings associated with process and environment improvements. Expected returns on investments will be calculated in accordance with guidelines worked out in cooperation with the STARS community.

A preliminary ROI figure using savings and value estimates derived from a pilot use of megaprogramming technology was computed[1]. The pilot demonstration (conducted between October 1992 and October 1993) resulted in the development of 125K physical lines of Ada software after about $2.9M was invested in the Demonstration Project. Using pre-megaprogramming models for the value of a line of code, we computed that the equivalent cost to develop such a product would be at least $12M, assuming a baseline cost of about $100 per line. (A more conservative valuation, which uses a baseline cost of $60 per line would be about $7.5M.) Since this is only a pilot demonstration of the development approach we will need to wait for actual software releases to obtain better cost comparisons. The current size estimate for Release 1 is 331K lines of Ada and the estimated total SCAI project cost at the time of delivery is only $6.3M. Using $100 per line for comparison, 331K lines would have cost over $33M, while using the more conservative figure of $60 per line still implies a value received of over $19M. If these numbers are sustained, and if the resultant software is of the same quality or better than software developed using customary SWSC procedures, a positive ROI has already been achieved on the SCAI project.

The error in this approach is setting the zero-point for the tracking of costs and values to be at the beginning of the SCAI demonstration program but using a baseline cost per line value which was only appropriate for non-megaprogramming, non-reuse SWSC development. In truth, the SCAI demo organization, SMX, had already made advances in reuse and megaprogramming prior to the beginning of the STARS involvement. Therefore, either the baseline cost per line should be set to the high reuse model appropriate for SMX prior to STARS investments or the prior technology investments should be included in the total costs (investments) prior to the Release 1 results. This latter approach would mean moving the zero-point for cost accumulation to a point before the development of the RICC reuse tools (e.g., Display Builder and Message Builder) and other reuse aids such as NAS and UNAS. In other words, the cost of development of such tools should be included in the total cost calculation, and the values already received from them in the form of products delivered prior to the Demonstration Project should likewise be included in the total value calculation.

After evaluating the pros and cons of both solutions, our preferred approach is to add the cost of the prior megaprogramming investments to the Demonstration Project costs, then to use the conventional development, non-reuse baseline cost for comparison. We prefer this approach because it captures all megaprogramming advances in both the cost and value parts of the total cost equation. Also, most other Cheyenne Mountain Upgrade (CMU) efforts still use conventional development approaches, so the use of the pre-reuse baseline cost is appropriate for depicting the potential savings from megaprogramming elsewhere in the CMU.

1. Using a model proposed by IDA.

Our estimate of the cost of the RICC tools is $2.7M, based on historical data.[1] Estimates of the costs of the other reuse tools are still being developed. Since the value of Release 1 will be about $33M (at $100 per line, although some estimates of the cost of space software are as high as $150 per line) we can absorb as much as $24M of other investments and still break even at the delivery of Release 1. This was computed by adding the known investments of $6.3M for the Demonstration Project to the $2.7M for RICC tools and subtracting from $33M as the value received from Release 1. Also, since there have been previous products which were delivered using these tools, the cumulative value received will have to be updated to include those outputs. The costs and received values of NAS and UNAS are still being investigated.

Until further data is available, a lower bound to current productivity can be established by considering only the newly developed code as output and dividing by the cost of the release. In this way, we eliminate any need to add the cost of reuse tools since we are not taking any credit for the reused code which is generated by those tools. Since the cost of the release covers all Demonstration Project activities, this is clearly a superset of the actual cost to develop the new code. Out of the total 331K lines of Ada comprising Release 1, 115K lines were new code. Dividing this into the cost incurred during the release (after the pilot), which is $3.4M, reveals that the cost per line for the release can be no higher than $30. This is well below all known estimates of the baseline cost of developing SWSC software.

Data Collection:

We are collecting labor hour expenditures by our work breakdown structure. Contractor leads and Air Force personnel have the ability to record effort daily using Amadeus. Results of effort by activity are reported on a month-by-month basis. We are collecting labor hour expenditures by our work breakdown structure.

Initially, effort data was collected manually, stored in Framemaker files, and imported into Amadeus for graphing. As the team became more familiar with Amadeus, a pop-up input window was developed so that project personnel could directly input their daily effort. The input window appears as part of the default login procedure and can be dismissed by the user if no hours need to be reported.

The metrics team has met with the development coordinators to plan breaking out a further granularity of effort reporting. Our goal is to obtain effort measurements for each developed unit of code in order to eventually relate development effort to other factors such as defect density and rework. The Process Management support tools will allow us to track effort with respect to defined tasks, such as the development of a particular unit of software.

We have some defect data with respect to the development of the SCAI SEE, but the metrics team has not yet collected defect data on Release 1 or Release 2 software.

The development team keeps defect logs for all Release 1 software and tracks all issues concerning Release 2 software beginning with the formal inspection of any unit. The metrics team will participate in the use and analysis of this data during the next part of the Performance Phase.

We have developed the mechanisms using Amadeus to collect tool usage statistics but have not yet implemented them on every SCAI SEE platform. This effort gives us the ability to predict what tools are needed where, and for how long. Also, we can determine the correct numbers of licenses required to support a project of our size. We do have reporting capability. However, in order to produce meaningful output we must amass several months of data.

Actual sizes of software units (Ada compilation units comprising software classes) are being collected and reported, but we have not begun collecting size estimates.

The estimated high-level project schedule is documented in project briefings. We are not yet tracking the actual or estimated project schedule in greater detail. Automated process tools will make this job straightforward.

We have given three user surveys. Two surveys, a year apart, have been given to users of the SEE, and one Technology Transition Survey was given to evaluate how technologies are being accepted.

1. Conversation with Lt. Col. Dave Bristow, who was the SMX Program Manager during the RICC tools development.

5.4.4 Lessons Learned

Lesson: *There must be a demand for the visibility that metrics can provide into processes or products.*

Demand for metrics is critical for a truly useful collection of project metrics data. This demand might come from management wanting visibility into process workings or product quality, or from each project participant wanting to better understand how they individually or the project in general is doing. On the SCAI project, the personnel assigned to metrics were also responsible for several other activities which required producing deliverable products, and given that there was no formal demand for reporting metric results, metric activities often took the back burner. If the project is not experienced with developing project goals for process and product performance, and collecting data to determine and report status in relationship to those goals, a full time metrics coordinator could be responsible for generating the demand for metrics. This coordinator would need visibility into all the project workings from the start to instrument the processes for metrics collection and reporting. The metrics coordinator needs to create a demand for metric results and establish a plan detailing how metrics data will be used, and prototype the presentation of project results. Management buy-in is a must. Coordinate the direction and uses of the data with management. Train the practitioners in the culture and use of metrics. Make the presentation of metric results a normal part of regular project reviews and/or daily business. Reporting of metric analysis must come to be viewed as business as usual if process and product improvement based on metric information collected and analyzed is to survive.

Lesson: *Metrics definition should be a part of process definition.*

It is impossible to identify metrics or to define a metrics collection process until the process being measured has been defined. The simultaneous consideration of process definition and measurement of the process and its products is the best way to ensure that metrics are feasible and meaningful. When process performers have some idea of the current state of their performance it is easier to identify the goals that they would like to meet, and once meaningful targets have been established, metrics to judge performance in relationship to targets is feasible. On the SCAI project high level project goals were established during the Preparation Phase, but definition of specific measurements could only take place after processes were defined. However, it was difficult to get people to set specific process or project goals, as the project had little to no history data on similar process performance, and no familiarity with producing the SCAI process products or using the SCAI process.

Lesson: *Organizationally-imposed metrics may require extra work but can provide a means to transfer new ideas to the organization.*

Participation in the SWSC Metrics IPT has benefitted the SCAI project by promoting uniform reporting of core software measurements across the SWSC which will assist the SCAI project in evaluating the effectiveness of its technologies. The other SWSC participants also benefitted from the goal-directed metrics definition that was used by the SCAI project to ensure that the measures in its core set were meaningful and valuable.

Lesson: *The business case for megaprogramming investments (e.g., process definition, reusable tools) should include return on investment (ROI) calculations.*

Although the figures presented here are encouraging in terms of the value received so far for megaprogramming investments, one should be careful to compare the up-front costs of a new technology with the value to be received over the life of that investment. It may be that several applications of a tool or process will be required before a net benefit can be derived. The use of ROI analyses can reveal the break-even/improvement point for such investments.

Lesson: *ROI business case development can be complicated.*

There are several pitfalls in the use of ROI to reflect the value received from a technology. We discovered many different approaches to computing ROI and selected the method illustrated above because of its relationship to a more standard economic measure, net present value (NPV). However, even this simplified approach to computing ROI led to difficulties when trying to establish a common baseline from which to compute total return and total investment.

5.4.5 Summary

During the first half of the Performance Phase the SCAI metrics team fulfilled its goals of implementing goal-directed measurement of the Demonstration Project. Effort by major activity and problem tracking relative to the SEE were implemented. Mechanisms for further metrics collection were developed and are being tested before being implemented across the project. These include effort by software unit, tool usage, and software unit size. Three Metrics Reports were published to record our progress during the year. The major lessons learned include the need to motivate the demand for metrics at all levels of an organization. Management must demand visibility into the process and products. Feedback to developers must be provided to summarize the quality and quantity of their work and to provide incentive for improvement. A full-time metrics coordinator is required to keep the process moving forward. Additionally, the team gained experience developing a business case for new technologies and computing a return on investment. Further work in all these areas is planned.

6.0 Technology Transition

6.1 Introduction

The ultimate purpose of the Air Force STARS Demonstration Project is to transfer megaprogramming technologies throughout the Air Force and the rest of the DoD community. The SCAI project's mission is to formalize the megaprogramming approach, demonstrate its effectiveness and transition the technology. There are three aspects of technology transition to be considered;

(1) Transition of technology into the SCAI project in order to demonstrate its potential and gain experience for further transition.

(2) Transition of technology out of the SCAI project, in a focused way, to an identified set of technology "customers."

(3) Providing awareness of the technology to a wider DoD community via technical papers, demonstrations, and participation in conferences.

There are currently 12 different technologies that are being transitioned into the SCAI project to provide the needed support for megaprogramming. The first customers for the transition of the technology out of the SCAI project are the Space Warning Systems Center and US Space Command. This section summarizes the experience gained related to technology transfer.

6.2 Long Term Objectives

The long term Technology Transition objectives are covered in the third project goal (see section 1 for a description of the project goals): "Initiate the institutionalization of the SWSC megaprogramming paradigm used to develop the SCAI." The intent is to leverage the success of the SCAI project and the experience gained to accelerate the shift to a megaprogramming paradigm. The three key objectives under this goal are:

(1) Demonstrate widespread dissemination of technological expertise across the SWSC organization and its contractors.

(2) Establish a product-line organization and infrastructure managing the evolution of multiple application systems.

(3) Transition technology/assets to other organizations for their own pilot application.

SCAI Project Technology Transition Strategy

Each of the three aspects of transition described above require a different strategy.

Transition into the SCAI Project: During the Preparation Phase and the first year of the Performance Phase the emphasis has been on transitioning the technology into the project. A complicating factor is the number of different organizations involved: approximately 50 people from 11 organizations (Air Force and 10 contractors) participating on the project. The staff must absorb the many technologies, adapt them to meet the projects needs and finally integrate all of these technologies into a common way of doing business. It was not possible to have a gradual ramp up or staging of the technologies. By the time the project personnel were in place, the application engineering, domain engineering and process definition were already underway. The project therefore had to deal with introducing, tailoring and integrating these technologies in parallel with other project activities. The strategy was to provide technology advocates/coaches as members of the project team to provide formal training, on the job coaching, tailoring and integration of the technologies.

Transition out of the SCAI Project: The experience gained from the transition of these technologies into use on the SCAI project will help to accomplish the SCAI project technology transition objective. The insight gained relative to the obstacles, risks and benefits will be used in defining the transition plan to the next set of customers. The strategy will be based on a number of transition models:

(1) Commitment Model (S curve): this model addresses a series of states an organization undergoes during transition (contact, awareness, understanding, trial use, adoption, institutionalization). [Conner 82]

(2) Role Model: this model describes the roles played by the sponsor (producers), champion (advocate), agent (receptor), target (consumer). [Fowler 88]

(3) Layered Behavioral Model: this model describes interactions and behaviors between people, teams, projects, organizations, and businesses. [Curtis 88]

(4) Crossing the Chasm: this model addresses customer analysis, their needs, and products. It suggests a marketing approach to selling the technology and concentrates on the problem of transitioning from technologists/visionaries to pragmatists. [Moore 92]

Widespread Awareness: The SCAI project has limited resources for spreading the technologies throughout the Air Force and DoD community. The project strategy is to generate technical papers based on experience. The information from this experience report is being disseminated in this way. Capturing and formally documenting experience aids technology transition as well as process improvement. The project has also selected one major conference each year (STC conference) for focusing attention using a demonstration booth, papers and panel participation.

6.3 Preparation Phase

The first version of this experience report, produced after the Preparation Phase, did not contain a section on Technology Transition. The actual transition of the technology into the project had just begun at that time and little concrete experience was available.

6.4 Performance Phase

6.4.1 Plans

Objectives Through the First Year

The initial objective was to have all the technologies available for use by the start of the Performance Phase (October '93), with training to be provided just prior to the first use of each technology. By the end of the first year of the Performance Phase the project personnel should have gained reasonable familiarity and **practical experience** with the technologies. More important was the objective of having buy-in by the project and the project members becoming **proponents** of the technology.

Success with the above goal by the end of the first year of the Performance Phase will allow attention to be focused on transitioning the technology outside the project. There was no objective to have the technology picked up outside the project prior to the first year, therefore this section of the experience report will focus on the transition of technology into the project.

Planned Activities

The plans for transitioning technologies into the project were:

(1) Make the technology ready for use by project personnel by October '93.

(2) Develop training approaches tailored to the needs of the users.

(3) Tailor the technologies based on the users needs.

(4) Integrate the technologies where required.

(5) Provide in-house coaching on the technology to assist its users in learning, tailoring and integrating.

Even though our objective was only to transition the technologies into the SCAI project, we did participate in the STC '94 conference by providing awareness of the SCAI project technologies in the SCAI and STARS booths. This was intended to prepare us for future transition activities to the larger DoD community.

6.4.2 Summary of Accomplishments

During the first year and a half, there was a constant focus on the transition of a number of new technologies into use on the SCAI project. Some were very mature; others turned out to be far less mature than expected. The technologies were installed or otherwise made ready for use by October '93 with the exception of the process support tools which were delayed until January '94. Some level of training was provided on each technology. Some of the technologies required significant tailoring and additional tailoring still needs to be done on some technologies. The integration of several of the technologies took longer than originally anticipated.

A number of papers were presented at conferences including STC '94 which supported the objective of spreading awareness of the technologies. Looking ahead to transition out of the SCAI project, a customer analysis was completed and the project became actively involved in the SEPG for the parent organization (SWSC).

6.4.3 Analysis

This section of the report is intended to analyze the project experience with respect to transition of technologies and will not address individual technologies specifically. The previous sections of this report also contain a number of transition lessons associated with the specific technologies. In particular, section 4.4.4 on SEE lessons learned breaks out a subsection related to introducing a significantly new SEE approach into an organization.

During October '94 a survey was conducted to evaluate the progress of the transition for 12 new technologies being used on the SCAI project. Several major technologies were not covered including Rational APEX (not considered a new technology for the project), Rational SoDA and ProDAT (SoDA and ProDAT are only in trial use at this point). The following table summarizes the responses received to the questions relative to the improvement in the users' experience level with the technology and the shift in the users' opinion regarding the usefulness of the technology.

As you can see by the table, progress was made in both the experience and opinion. Progress, however, was not as significant as planned. The objective was to get the project personnel to an experience level of "practical experience" (3 on experience scale). Although significant progress was made, it fell short of that goal. Similarly in the area of opinion, the objective was to bring the staff to a level of proponent (4 on the opinion scale) and again it fell short of that objective. The survey contained a number of other questions which solicited comments about the transition. The survey as well as additional experience information collected from the project staff formed the basis of the lessons learned contained in this report.

Technology	Number of Responses	Experience Shift 10/93 to 10/94	Delta	Opinion Shift 10/93 to 10/94	Delta
OO/Booch	7	1.4 to 2.2	0.8	2.9 to 3.7	0.8
Cleanroom	8	1.0 to 2.0	1.0	3.0 to 3.7	0.7
Ada Process Model	7	1.7 to 2.6	0.9	3.0 to 3.6	0.6
RICC	8	1.0 to 2.2	1.2	2.4 to 3.8	1.4
IDEF	9	1.6 to 3.3	1.7	3.0 to 3.0	0.0
ETVX	5	1.3 to 2.5	0.8	2.8 to 3.3	0.5
Info Organizer Templates	8	1.1 to 2.8	1.7	3.0 to 3.0	0.0
SPMS	8	1.0 to 2.5	1.5	2.7 to 3.3	0.6
Catalyst	6	1.1 to 3.4	2.3	2.7 to 3.4	0.7
Process Weaver	6	1.1 to 2.6	1.5	3.0 to 3.5	0.5
Amadeus	10	1.0 to 2.6	1.6	3.1 to 4.1	1.0
CMVC	14	1.2 to 1.8	0.6	3.3 to 3.5	0.2
Megaprogramming	13	1.7 to 2.9	1.2	3.5 to 4.3	0.8
Average Shift		**1.3 to 2.6**	**1.3**	**3.0 to 3.6**	**0.6**

KEY:

Shift: Change from when the project started on 10/93 until now, 10/94
Experience Shift: How experienced were you (are you) with using this technology?
1=no direct, 2=limited, 3=practical, 4=extensive, 5=could teach
Opinion Shift: How did you (do you) feel about using this technology?
1= strong opponent, 2=opponent, 3=neutral, 4=proponent, 5=strong proponent

Table 8. Technology Transition Survey Results Summary

So the question is: what happened during this period of about a year and a half? We started out with a powerful set of new technologies and positive attitudes by all the project members to try to make a successful transition to everyday use. We fell short of our goal for a number of reasons.

We were not realistic about the training cost and impact. A person's ability to become proficient in the use of a technology is much more a function of practical experience than training. We attempted to plan a large number of training classes, many of which never took place due to the logistics and the overall impact they would have caused the project. But even if we had conducted all the training courses, people would still have required many months of hands on experience. In other words, technology proficiency takes time and you need to temper your expectations because of this. If we were to do this again, we would focus far less on formal training sessions and far more on coaching by technology experts within the team and training materials which could be used by individuals when they had available time.

We also had an ambitious project schedule which quickly began to consume the project resources. Several of the technologies were still being brought on-line as the project moved into the Performance Phase. This limited the ability of the project personnel to understand how to best apply the technology and took away opportunities for more training and evaluation of the technologies. Its unlikely that any project will be able to avoid this conflict but your expectations should be set with the understanding that in most cases, the first use of a new technology will not immediately result in improvement.

One major thing we should have done differently was to better anticipate the difficulties in the transition of such a large number of technologies and build reflection phases into our project schedule. We tend to be culturally and organizationally driven by schedule and commitments. One way to deal with this is to emphasize a commitment to "institutionalizing" the use of new technologies by building reflection and improvement cycles into the project plan. Good intentions will generally take a back seat to stated commitments. So it is critical to define a stated commitment to the transition of new technologies and track progress in the same way we track product progress.

All things considered, the SCAI project has progressed extremely well with the transition. The success we have had is mainly attributable to the attitudes and efforts of the individual team members. As you read the following lessons it is important to understand one aspect of the SCAI project that was a major contributor to the success we have had in transitioning technologies into use. We had a willing, enthusiastic group of people from the start. The project was formed in order to demonstrate the benefits of new technologies. The team accepted that as a challenge from the beginning. We did not have to overcome the initial organizational inertia and spend large amounts of time gaining buy-in. The project members understood that the project goal was to demonstrate a new way of doing business faster, better, cheaper than was possible in the past. The SCAI team came predisposed to trying a different approach. If your organization is not predisposed, much more ground work will be needed to motivate the team.

6.4.4 Lessons Learned

If you are experienced with technology transition, many of the lessons that follow here will sound familiar. Prior to the start of the SCAI project, there had been a significant focus on the problems of transition. The team had previous experience and training in technology transition. However, there is a tremendous urge to try to push technology into use as fast as possible and a temptation to take transition risks. A number of the lessons below are a result of just such risks.

The Technology Transition lessons are divided into 3 subcategories:

(1) Technology Transition into the SCAI Project

(2) Technology Transition out of the SCAI Project

(3) Widespread Awareness

This version of the experience report will address items 1 and 3. The final version will cover transition out of the SCAI project.

Technology Transition into the SCAI Project

Lesson: *A technology transition plan is critical to success.*

The SCAI project was essentially a technology transition project with a goal of demonstrating an improved way of building real DoD systems using a set of new technologies supporting the concept of megaprogramming. There was no aspect of the SCAI project that did not involve the application of new technology. The use of new technology was so pervasive across the project that it was assumed that the detailed planning for the project would adequately handle the transition aspects. Although transition of new technology was certainly addressed in the detailed plans, the overall impact was underestimated and the interaction between the technologies was not well understood. Examples of this show up in the lessons below. Risks are often understood and accepted by the project team, but without a single, documented plan to address transition and its risks, the overall impact to the project will likely be underestimated.

Lesson: *Transition of multiple technologies, not previously integrated, is very difficult.*

The amount of change that the team can reasonably absorb and the integration of multiple technologies should be considered for the group of technologies rather than each technology alone.

The SCAI project objective was not just to build a system but to use a number of new technologies while building the system in order to demonstrate the potential of megaprogramming. Because of this, a significant amount of resource was allocated to the project by both STARS and the Air Force to support the transition and adaptation of the technologies. There was no opportunity to stage the transition of the technologies and minimize or control the impact of changing the short project time frame. This is not

something that should be attempted without significant additional resources and a strong desire to remake your organization. At least 20% of the resources going into the SCAI project have been used for transition related activities such as training, coaching, maturing/tuning/integrating technology and creating a process which ties the technology into a new way of doing business.

The complexity of the integration of multiple new technologies should not be underestimated. One specific problem is the difficulty of assessing the impact of a single technology (either positive or negative). Another is the impact of a massive learning curve on the project team with little extra time added to the project schedule to compensate for hands on familiarization with real use.

Lesson: *The more organizations involved in interactively applying the technologies, the more difficult the transition will be.*

The SCAI project team is led by the Air Force and includes 10 different contractor organizations. This creates a major obstacle to transitioning new technology. Successful transition of technology requires close attention to the needs, background, experience level, and organizational culture of the users of the technology. The approach to the transition must be tailored to these factors. In the case of the SCAI project, this becomes almost impossible because of the number of organizations involved. The communication paths are more restricted than you would find in a single organization. Gaining team buy in is much more difficult in these circumstances.

Lesson: *New technology may bring unanticipated baggage that needs to be controlled and maintained.*

All software projects are faced with a considerable amount of cost associated with maintaining intermediate artifacts and multiple versions. When you introduce new technology into your process you must consider the impact of maintaining and controlling additional artifacts. The SCAI project has many examples of this. A prime example is process definition. Process is very much like software. When you define it you create a number of intermediate artifacts such as models, guides and enactment scripts. All of these new artifacts need to be maintained and kept in synchronization. In the case of new technology, you will often find you need new tools which are not integrated together nor integrated with your existing toolset (complicating the problem of maintenance). The SCAI process definition is stored in multiple data bases making the update activity difficult.

Lesson: *Parallel definition of the project's process with the introduction of new technologies presents the classic chicken and egg problem.*

A key tenet of Process-Driven Development is that you should let the needs of your process drive your technology decisions. This is a very important aspect of process improvement. However, when you decide to make major technology changes either because of process needs or the desire to change the paradigm you have been using, you will be faced with which comes first. You must modify your process to incorporate the new technology and at the same time try to gain experience with the technology so you will best understand how to incorporate it into your process. Two major activities of the SCAI project were to formally define the process and at the same time apply many new technologies in that process. As a result there was need for a great deal of iteration. It also turned out to be much more difficult than anticipated to define the process because of the interaction required between the technology experts and the process users. On the other hand, by formally defining a process which incorporated the new technology, many problems were uncovered early in the project.

Lesson: *Delay the application of immature or unintegrated technology until after successfully piloting it with user involvement.*

When you are introducing new technology you will generally be dealing with some level of immaturity. The process tools being applied by the SCAI project to automate the definition and use of process turned out to be too immature for use at the beginning of the project. Because of a late delivery (typical of immature technology) there was no time to conduct a pilot application of the technology prior to installing them for use by the development team. As a result, all the expected things happened. The tools had bugs, some of the capabilities were not available, the users had no previous experience in using the tools, training was rushed, and some of the features of the tools turned out to be unusable because they were not integrated with other parts of the environment. A small pilot application of these tools would have uncovered these issues with minimum impact to the rest of the project. The cost of pushing these tools into use before they were ready was two fold. First, a great deal of time was spent responding to and working around prob-

lems. This time would have been better spent improving the tools. Second, the users confidence was destroyed, making the second attempt at transitioning them into use more difficult.

Lesson: *Put together a realistic training plan.*

It is important to find the correct balance between the needs for extensive training on new technologies and the practical need to begin work and gain experience with the technology. For example, an extensive training plan was developed for all 12 new technologies being applied to the SCAI project, but it did not deal with the issues of tailoring the technologies or integration of multiple technologies. It was also unrealistic in terms of its impact on the project team. One specific example was a Cleanroom class which was built assuming two week training. This class had to be cut at the last minute to a 1 week class. The class addressed pure Cleanroom and had to limit the amount of practical hands-on workshop sessions. The technology transition survey taken after the first year of use indicated two problems: 1) the class taught pure Cleanroom and not the modified form of Cleanroom that was used, and 2) users did not have enough practical experience on Cleanroom when they needed to start their work. Training plans need to incorporate some amount of iteration where you learn the basics of the technology, tailor it to the project needs, gain some experience and then do additional training based on the way the project will apply the technology.

Lesson: *Just in time training is a good approach but is difficult to achieve.*

The best time for training on a new technology is just prior to when the team needs to apply it. However, new technology requires an iterative training approach as described in the above lesson. Just-in-time also means that training is constantly competing for the time of the team members throughout the performance of the project. The SCAI project was forced to cancel or drastically limit training time because of resource availability. Another problem is developing training for the new personnel joining the project after start up. Training tended to rely too heavily on formal classroom methods and not enough on guidebooks, examples and tutorials. The use of small technology pilots can also help mitigate some of the problems by giving some team members early, practical experience and then having them help other team members during the actual application of the technology.

Lesson: *Formal training is no substitute for in-house technology coaching by technology experts.*

The SCAI project made use of technology experts/advocates to support the transition of the technologies. These people were part of the team and were responsible for training, and day-to-day coaching. By being part of the team they were able to help adapt the technology to the needs of the project.

Lesson: *Experience capture is a necessary part of Technology Transition.*

The requirements of the STARS program make it mandatory that the project capture its experience in applying megaprogramming. This forcing function makes the project take stock of the transition progress and improve the application of the technology. This needs to be part of your overall project plan or it will never get the priority necessary when competing for valuable resources. This is an essential part of a Plan/Do/Learn paradigm (as described in the STARS CFRP).

Widespread Awareness

Lesson: *Demonstrating and documenting the experience with the use of new technology builds common understanding and buy-in among the project staff.*

The SCAI project has done a large number of demonstrations of the technology and has developed a number of jointly authored papers based on this experience. This is not only beneficial to the outside community, it has also done a lot toward improving the project's common understanding and acceptance of the technology.

6.4.5 Summary

The transition of many new technologies into a project made up of the Air Force and 10 different contractors has proved to be a challenging task. The lessons learned include the need for a formal technology transition plan, updated based on experience. Avoid transitioning multiple technologies simultaneously, but if you must do this expect additional complications with technology integration. Use an iterative approach to technology transition and process improvement. Be realistic with your training plan in order to balance

the need for significant amounts of training with the need to get the project going and gain some experience with the technology. And finally, pilot the new technology with actual user involvement prior to release for operational use.

7.0 Conclusion

This report has provided an overview of the SCAI project approach and the experience gained during the Preparation Phase (October 1992 through October 1993), and the first year of the Performance Phase (November 1993 through October 1994). This report will be further updated early in 1996, at the completion of the project. Interim updates on the status of all the STARS Demonstration Projects will be provided in the STARS newsletter.

We hope this information is of value to your organization, and we welcome any feedback or comments.

Appendix A

Acronyms and Definitions

Acronyms

1CACS	1st Command and Control Squadron
AAM	Application Architecture Model
ABC	Activity Based Costing
ACS	Ada Compilation System
ADL	Ada Design Language
AE	Application Engineering
AF	Air Force
AFB	Air Force Base
AFC^4A	Air Force Command, Control, Communications and Computers Agency
AFMC	Air Force Materiel Command
AFSPACECOM	Air Force Space Command (replaced by AFSPC acronym)
AFSPC	Air Force Space Command
AIX	Advanced Interactive eXecutive (IBM UNIX operating system)
AJPO	Ada Joint Program Office
ALM	Application Logical Model
AMS	Assets Management System
API	Application Programming Interface
APM	Assistant Project Manager
APP	Application Portability Profile
APSE	Ada Programming Support Environment
ARM	Application Requirements Model
ARPA	Advanced Research Projects Agency
ASDDPP	AF/STARS Demonstration Detailed Project Plan
AWG	Architecture Working Group
BMS	Background Message Services
BPIP	Business Process Improvement Program
BPS	Bits Per Second
C^2	Command and Control
C^2AI	Command and Control Architecture Infrastructure
C^3	Command, Control and Communications
C^3I	Command, Control, Communications and Intelligence
C^4I	Command, Control, Communications, Computer and Intelligence

CASE	Computer Aided Software Engineering
CC	Commander
CCE	Squadron Section Commander
CCF	First Sergeant
CCO	Common Communications (CSCI)
CCPDS-R	Command Center Processing and Display System Replacement
ccPE	Cedar Creek Process Engineering
CDE	Common Desktop Environment
CDR	Critical Design Review
CDRL	Contract Data Requirements List Critical Design Review List
CDW	Critical Design Walkthrough
CFRP	Conceptual Framework for Reuse Processes
CIM	Center for Information Management Corporate Information Management
CINC	Commander in Chief
CM	Configuration Management
CMAFB	Cheyenne Mountain Air Force Base
CMAH	CINC Mobile Alternate Headquarters
CMAS	Cheyenne Mountain Air Force Station
CMC	Cheyenne Mountain Complex
CMM	Capability Maturity Model
CMP	Common Mission Processing (CSCI)
CMU	Cheyenne Mountain Upgrade
CMVC	Configuration Management Version Control
COM	Computer Operation Manual
COSE	Common Open System Environment
COTS	Commercial-Off-The-Shelf
CPM	Computer Programming Manual
CR	Cleanroom
CSC	Computer Systems Components Computer Software Component
CSCI	Computer Software Configuration Item
C/SCSC	Cost/Schedule Control System Criteria
CSO	Computer System Operator
DA	Domain Analysis
DAM	Domain Architecture Model
DAPM	Domain Analysis Process Model
DARPA	Defense Advanced Research Projects Agency (renamed ARPA)
DASD	Deputy Assistant Secretary of Defense

DBDD	Data Base Design Description
DBMS	Data Base Management System
DCO	Display Coordination (CSCI)
DDRS	Defense Data Repository System
DE	Domain Engineering
DEC	Digital Equipment Corporation
DID	Data Item Description
DISA	Defense Information System Agency
DJAG	Demonstration Joint Activity Group
DLM	Domain Logical Model
DoD	Department of Defense
DPM	Deputy Program Manager
DRM	Domain Requirements Model
E-Mail	Electronic Mail
EIL	Elaborate Infrastructure Layer
ERD	Entity Relationship Diagram
ERM	Error Monitor
ESC	Electronic Systems Center
ESIP	Embedded Computer Resource Support Improvement Program
EST	Engineering String Test
ETVX	Entry, Task, Validation, eXit
FCA	Functional Configuration Audit
FEA	Functional Economic Analysis
FFRDC	Federally Funded Research and Development Center
FODA	Feature Oriented Domain Analysis
FPI	Functional Process Improvement
FQT	Formal Qualification Test
FSC	Federal Systems Company
FS-G	Federal Systems - Gaithersburg
FSM	Firmware Support Manual
FTE	Full Time Equivalent
FTP	File Transfer Protocol
FY	Fiscal Year
GAC	Generic Application Controller
GCCS	Global Command and Control Systems
GFE	Government Furnished Equipment
GMP	Generic Message Parser
GOTS	Government-off-the-Shelf
GQM	Goal/Question/Metrics
HMI	Human Machine Interface
HP	Hewlett Packard Corporation

HQ	Headquarters
HW	Hardware
IBM	International Business Machines Corporation
ICAM	Integrated Computer Aided Manufacturing
ICOM	Inputs/Controls/Outputs/Mechanisms
IDA	Institute for Defense Analyses
IDD	Interface Design Description
IDEF	ICAM Definition
$IDEF_0$	IDEF -Activity Modeling Technique
$IDEF_{1x}$	IDEF - Rule Modeling - Data Model
IEEE	Institute of Electrical and Electronic Engineers
IM	Information Management
IPDR	In-Progress Preliminary Design Review
IPT	Integrated Process Team
IR	Incident Report
IRS	Interface Requirements Description
IR&D	Independent Research and Development
ITC	InterTask Communications
ITD	Inception to Date
ITW&AA	Integrated Tactical Warning and Attack Assessment
IV&V	Independent Validation and Verification
I&T	Integration and Test
KSC	Kaman Science Corporation
KSLOC	Thousand Source Lines of Code
LLCSC	Lower Level Computer Software Component
LOC	Lines of Code
MCCC	Mobile Consolidated Command Center
MCCR	Mission Critical Computer Resources
MCCS	Mobile Command and Control System
MIL-STD	Military Standard
MLOC	Million Lines of Code
MOA	Memorandum Of Agreement
MOD	Modification
MOIA	Mission Operation/Information Analysis
MSS	Mission Support Segment
MW	Missile Warning
NAS	Network Architecture Services (CSCI)
NASA	National Aeronautics & Space Administration
NFS	Network File System
NIST	National Institute of Standards and Technology
NMO	Network Management Operator

NORAD	North American Aerospace Defense Command
NPV	Net Present Value
OBS	Organizational Breakdown Structure
OCD	Operational Concept Description
OIW	Open Systems Environments Implementors' Workshop
OJT	On-the-Job Training
OO	Object Oriented
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OS	Operating System
OSD	Office of the Secretary of Defense Object Scenario Diagram
PC	Personal Computer
PDIOT	Process Definition Information Organizer Template
PDL	Program Design Language
PDR	Preliminary Design Review
PDS	Processing and Display Subsystem
PDW	Preliminary Design Walkthrough
PE	Process Engineering
PEAKS	Process Engineering and Analysis Kernel System
PM	Program Management
PMR	Performance Measurement Review Program Management Review
POSIX	Portable Operating System Interface
PRC	Planning Research Corporation
PSE	Process Support Environment
PSS	Process Support System
QA	Quality Assurance
QBT	Query Builder Tool
QE	Quality Evaluation
QP	Query Processor
QPR	Quarterly Program Reviews Quality Performance Requirements
R1	Release 1
R2	Release 2
R3	Release 3
R&D	Research and Development
RD	Recursive Design
RG	Robbins-Gioia
RHMI	Reusable Human Machine Interface
RICC	Reusable Integrated Command Center

RISC	Reduced Instruction Set Computer
ROI	Return on Investment
ROI(P)	Return on Investment (Preliminary)
RSMP	Reuse Strategy Model Prototype
RTM	Requirements Traceability Matrix
S&W	Space and Weather
SAC	Strategic Air Command
SAF/AQK	Secretary of the Air Force/Acquisition (Software)
SAIC	Science Applications International Corporation
SALE	Software Architect's Lifecycle Environment
SAS	Software Architecture Skeleton
SAT	Stand Alone Test
SC	Directorate of Communications and Computer Systems
SCAI	Space Command and Control Architectural Infrastructure
SCCB	Software Configuration Control Board
SCF	Software Change Form
SCOM	Software Center Operations Manual
SDD	Software Design Description
SDDD	Software Detailed Design Document
SDE	Software Development Environment
SDF	Software Development File
SDP	Software Development Plan
SDPMP	STARS Demonstration Project Management Plan
SDR	State Date Repository System Design Review
SEDD	System Engineering and Development Division (TRW)
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
SEPG	Software Engineering Process Group
SET	Software Engineering Technology, Inc.
SIOM	Software Input/Output Manual
SIP	Software Integration Plan
SLOC	Source Lines of Code
SMC	C^3 Systems Directorate
SMI	Intelligence Systems Directorate
SMQ	Quality Control Directorate
SMR	Resource Management Directorate
SMS	Space Systems Directorate
SMW	Warning Systems Directorate
SMX	Plans and Engineering Directorate

SMXE	Systems Engineering/Technical Support for SMX
SMXM	Project Management for SMX
SMXP	Product Line Manager for SMX
SOW	Statement of Work
SPADOC	Space Defense Operations Center
SPM	SubProject Manager
SPMS	Software Process Management System
SPR	Software Problem Report
SPS	Software Product Specification
SQEP	Software Quality Evaluation Plan
SQL	Standard Query Language
SRA	Systems Research and Applications Corporation
SRS	Software Requirements Specification Space and Warning Systems Director
SRS-2	Deputy Director of Space and Warning Systems
SRSC	Command, Control and Communications Division
SRSE	Squadron Section Commander
SRSM	Mission Software Support Division
SRSP	Software Engineering Process Group
SRSQ	Quality Control Division
SRSS	Space Systems Division
SRSW	Warning Systems Division
SRSX	Plans and Engineering Division
SSC	Space Surveillance Center
SSDD	System/Subsystem Design Description
SSPM	Software Standards and Procedures Manual
SSR	Software Specification Review
SSS	System Segment Specification Staff Summary Sheet
SSSGCCE	Squadron Section Commander
SSSGCCF	First Sergeant
STARS	Software Technology for Adaptable, Reliable Systems
STC	Software Technology Conference
STD	Software Test Description
STLDD	Software Top-Level Design Document
STP	Software Test Plan
STPR	Software Test Procedure
STR	Software Test Report
STrP	Software Transition Plan
STSC	Software Technology Support Center
SUM	Software User's Manual

SVD	Software Version Description
S/W SW	Software
SWENG	Software Engineering
SWSC	Space and Warning Systems Center (replaced by SWSD)
SWSD	Space and Warning Systems Directorate
SWTP	Software Test Plan
TAR	Turn Around Report
TAS	Test and Simulation (CSCI)
TBD	To Be Determined
TCP/IP	Transmission Control Protocol/Internet Protocol
TDF	Test Data File
TLCSC	Top-Level Computer Software Component
TOR	Turn Over Report
TPR	Technical Program Review
TRR	Test Readiness Review
TT	Technology Transition
TW/AA	Tactical Warning/Attack Assessment
UNAS	Universal Network Architecture Services
USAF	United States Air Force
USI	User-System Interface
USSPACECOM	United States Space Command
VADS	Verdix Ada Development System
VIEWS	Visual Engineering Workstation
VMS	VAX Virtual Monitor System
WBS	Work Breakdown Structure
WPM	Work Package Manager
WWW	World Wide Web

Definitions

(1) Software Engineering Environment

A "software engineering environment" (SEE) is a complex system including many software tools, hardware components, and network capabilities communicating through standard interfaces. A SEE provides automated technology support for the people developing systems and the processes they use to develop the systems. The integration mechanisms of the Loral STARS SEE are Cap Gemini's Process Weaver, IBM's WorkBench, and standard relational databases.

(2) Process Support Environment

A "process support environment" (PSE) represents the subset of the SEE that provides the infrastructure capabilities for supporting the development and execution of a project's "process support system." The Loral STARS SEE PSE includes Process Weaver, ProjectCatalyst, Software Process Management System (SPMS), and CAT Compass.

(3) Process Support System

A "process support system" (PSS) is the system of processes, practices, guidelines and methods that guide project personnel in the planning, specification, development and delivery of a target system. A PSS can be an entirely manual system, partially manual and partially automated, and if practical, entirely automated. A project's PSS created by the Loral STARS SEE PSE consists of SPMS process definitions and the SPMS database, ProjectCatalyst and Process Weaver project process definitions, and the project's management plan.

(4) Architecture

Two uses of the term architecture are appropriate:

- The methods used to construct a system - including the prescribed use of:
  - Components, such as hardware, software, files, and networking interconnections,
  - Interfaces, such as APIs and standard message protocols,
  - Tools, such as code generators to assist in using an API, and
  - Programming language templates.
- The actual as-built structure of a system constructed using the above approach.

  (See also Domain Architecture, Architecture Models)

Further notes on architecture:

- The word Architecture refers to the "solution space" rather than the "problem space".
- Note that in a megaprogramming product-line, architectural commonality is given a high priority - since it is viewed as a prerequisite for large amounts of domain-specific reuse.
- There is a difference between "architecture" and "models of the architecture." The architecture is how the system is built, and we use models to depict various views of

the architecture. When working out an architecture, one can use models as communication and analysis vehicles.

- There are numerous valid views of architecture, and many valid modeling approaches to depict those views. When working out an architectural modeling approach, a project must decide which particular views are important - given the domain, the technology, and the people doing the work. The various models will hopefully be consistent - and tool support for maintaining/checking this consistency may well be appropriate.

The following views are candidates for the SCAI:

- Composition - the building blocks approach: subsystems, CSCIs, CSCs. Further, there are both static and dynamic subviews of this.

- Abstraction - distinguishing among layers of abstraction in order to separate concerns and in order to allow people to work on portions of the design independently. The Shlaer-Mellor ideas of Recursive Design (RD) are an example. Static and dynamic subviews may be appropriate here also (e.g., these-called "bridges" in RD that specify how the higher levels interact with the lower levels).

- Construction Methodology - the approach for fabricating the software. This should be worked out in the context of composition and abstraction above, and might be very specific to the product-line. Subviews of this view might be a set of templates of legal Ada constructs, SALE, SAS and GACs, and RICC tools/APIs. Again, note that the various views should relate to each other but it is important to recognize that all of them coexist; they all can be essential to understanding, communication, and engineering.

- Architecture Model

  A set of views of architecture for an application (or a domain) that provides a basis for communication and analysis. Views can include depictions such as:

  - Interfaces and "building block" composition,
  - Layers of abstraction, and
  - Construction methodology.

- Domain Architecture Model (DAM)

  An Architecture Model for a family of related applications in a megaprogramming product-line; the DAM identifies:

  - Architectural commonalities, and
  - Methods for adapting the common aspects to specific applications in the product-line.

- Application Architecture Model (AAM):

  An Architecture Model for an application in a family of related applications in a megaprogramming product-line.

(5) Domain Terms

[Note: The motivation for attempting to define this set of terms is that there is much confusion in our communication, because there are so many different possible meanings for the term "domain." It seems clear that we should seldom use the term "domain" without some suitable modifier - such as "application domain" or "user interface domain."

- Domain

  A coherent body of technical discourse that has been selected to assist in reasoning about a system or family of systems.

- Application Domain

  A domain that is restricted to the inherent aspects of a problem as distinguished from the aspects that are peripheral or subsidiary (see Service Domain, e.g.). The purpose in clearly defining the boundaries between the Application Domain and other domains is to allow application domain experts to concentrate on those aspects of the problem that they are most interested in and best suited for.

- Service Domain

  A domain that is viewed as subsidiary to a higher domain. The purpose in separating out a service domain is to remove an area of concern from the discourse of the using domain, freeing the analysts of the using domain to concentrate on only the most pertinent aspects. For example, an Application Domain could be distinguished from a Message I/O domain.

- Subdomain

  Usage of this term is discouraged, since its meaning is so prone to misunderstanding!

- Application Domain Family

  A family of related applications being analyzed as a group due to their similar characteristics; the motivation for treating them as a family is to identify and exploit commonalities among the applications so as to maximize "domain-specific" reuse.

  For example, in the SWSC we are interested in the Space and Warning Domain Family.

- Application Domain Subfamily

  A subset of the applications in an Application Domain Family that has been called out because of their strong similarities, allowing focus by specific application experts and giving rise to the opportunity for focused reuse. In a megaprogramming product-line, the system solutions for a Subfamily would take advantage of Domain Family reuse, but would tailor/adapt the Family's assets in a coherent way across the Subfamily - and they would also take advantage of assets common to the Subfamily, but not used in other Subfamilies.

  For example, in the SWSC we are interested in the "Space" Domain Subfamily of the "Space and Warning" Family.

- Product-Line

  A set of applications developed and maintained using a megaprogramming paradigm - i.e., using a process-driven, domain-specific reuse based, technology supported approach. An essential characteristic of a product-line organization is the presence of a focal point "Domain Manager," responsible for a domain family of applications, as well as the resources and processes used to develop and maintain them.

  The Domain Manager may have ownership of other applications in the Domain Family that are legacy systems and are not considered part of the product-line. The primary objective of transitioning to a product-line approach is to exploit the

commonality among the applications and to realize a high degree of systematic reuse.

- Domain Analysis

  The process of systematically analyzing a system or family of systems by dividing the universe of discourse into separate domains with well-defined interfaces. The purposes of such divisions may include: increasing the ability to attain intellectual control, separating out common aspects from unique aspects within a family of systems, and arranging for good encapsulation in system implementations.

  The term is used to characterize "problem" analysis; Domain Analysis is intended to provide artifacts that clearly characterize the problem to be solved and that serve to enhance the "solution" process.

  Although the definition allows Domain Analysis to take place for a *single* system, this is really just a special case of a *family* of systems. In other words, even while analyzing a single system, the analysis method attempts to focus on the essence of the problem and strip out most implementation considerations. Thus, the analysis work products should be viewed as supporting a family of application solutions.

  Possible meanings of the term "domain analysis "include:

  - Analysis of the inherent aspects of a single application, avoiding implementation-specific aspects (e.g., Ward. Mellor, McManamon/Palmer).
  - Analysis that involves separating the application into multiple parts, each of which is subject to delegated analysis by personnel with expertise in specialized disciplines, such as database management or operating systems (e.g., Shlaer/ Mellor).
  - Analysis of several related applications with the intent of flushing out and exploiting commonalities (e.g., Prieto-Diaz, Cohen).

(6) Engineering Terms

- Application Engineering

  Engineering (specifying, developing, deploying, and maintaining) a software-based system solution for an application in accordance with a megaprogramming product-line strategy that applies to multiple similar applications. (See also Domain Engineering.)

- Domain Engineering

  Engineering a family of similar applications in accordance with a megaprogramming product-line strategy, providing a common architectural approach, tools, domain models, and other reusable artifacts for the entire family. (See also Application Engineering.)

- Database Engineering

  Team responsible for engineering the common state data used for an application or family of applications. This may involve modeling the data, developing schemas, defining data dictionaries, and designing queries.

- Distributed Processing Requirements

  Allocation of the capabilities of a system to nodes in a network environment and specifications of their interface and performance characteristics.

(7) RICC - Reusable Integrated Command Center

A generic implementation of a Space and Missile Warning Command Center based on a reusable architectural infrastructure and an associated tool set. The first instantiation of the RICC was a pilot in the Missile Warning domain.

- RICC Architectural Infrastructure

  An open systems-based set of APIs and Ada run-time components that were first used to build the RICC MW pilot, and which are being used to build the SCAI application. These consist of the Reusable Human Machine Interface (RHMI) for interactive displays, the Generic Message Parser (GMP) for message handling, and the Query Processor (QP) for database operations. The infrastructure also includes the Universal Network Architecture Services (UNAS) for the run-time network executive. (See also RICC and RICC Tools)

- RICC Tools

  A set of interactive tools used to generate code to the RICC architectural infrastructure APIs. These consist of Display Builder (DPT - for RHMI), Message Tool (XMT - for GMP), Query Builder Tool (QBT - for QP), and Software Architect's Lifecycle Environment (SALE - for UNAS). (See also RICC and RICC architectural infrastructure).

Appendix B

Technologies Contributing to SCAI

Background Technologies

This Appendix covers the technologies being used on the SCAI project that have influenced the project's technical approach. Many state-of-the-art incumbent technologies existed within the SWSC prior to the arrival of STARS technologies. Descriptions of the incumbent technologies, the STARS technologies, and a description of a technology for defining processes created as a result of a process affiliate relationship with the Software Engineering Institute (SEI) are provided.

(1) Technologies already in use at the SWSC at the start of the Demonstration Project (incumbent)

- Corporate Information Management (CIM) Process Initiative
- Reusable Integrated Command Center (RICC)
- Mission Operation/Information Analysis (MOIA)
- Booch Object Oriented Analysis
- TRW Ada Process Model

(2) STARS Technologies

- STARS Process-Driven Development
- Automated Process Support Tools
  - ProjectCatalyst
  - Process Weaver
  - Software Process Management System (SPMS)
  - CAT Compass
  - Amadeus
- Cleanroom Software Engineering Process
- Domain Analysis Process Model (DAPM)

(3) SCAI/STARS/SEI Process-Driven Technology

- Information Organizer Templates

Incumbent Technologies

Corporate Information Management (CIM) Process Initiative

The Deputy Assistant Secretary of Defense (DASD) for Information Management (IM) sponsored a Corporate Information Management (CIM) workshop of the processes performed in conducting software maintenance at the SWSC. DASD IM commissioned a Joint Services' CIM Working Group to conduct this workshop and improve the activities associated with the SWSC. The group conducted its modeling efforts

in accordance with DoD 8020.1-M, Functional Process Improvement Program, 1 October 1992. Participants included personnel familiar with the operation of the SWSC, the Integrated Tactical Warning/Attack Assessment (TW/AA) configuration control process, and operations within the Cheyenne Mountain Air Force Station (CMAS). The Integrated Computer Aided Manufacturing (ICAM) Definition (IDEF) techniques were employed to do the business process models.[1]

The CIM program is aimed at changing the way people work in the DoD so that the DoD operates with cost optimization and performance excellence objectives. To implement the CIM initiative, the DoD has created a Business Process Improvement Program (BPIP) to encourage a consistent application of process improvement principles and techniques across its services and agencies. The objectives of these techniques is to eliminate duplication of functions and the redundancy of business processes and information systems. The BPIP contains general concepts and steps associated with business process improvement and incorporates the development of a business case for evaluating investments.

"The DoD is using business process improvement workshops to identify inefficiencies, poor business practices, and costly non-value added activities. These workshops enable functional managers to identify current problems, establish costs for business activities, propose change alternatives, and implement business improvements in their organizations and business processes."[2]

The key objectives of the BPIP include:

(1) Building a model and establishing cost and performance measures of the baseline to enable the organization to demonstrate improvements.

(2) Identifying and eliminating non-value added activities.

(3) Simplifying, integrating, and streamlining value added activities.

(4) Emphasizing reuse of assets whenever possible.

(5) Automating only after underlying business processes have been cleaned up.

(6) Aligning goals, policies, and procedures within the CIM Integration Architecture (the reference model that guides all information systems implementation activities providing a strategic framework for making decisions that affect the DoD information infrastructure).

(7) Integrating processes, physical assets, organizations, and data as appropriate to gain economies of cumulative volume and limited redundancy.

To achieve these objectives the CIM Function Process Improvement Technology was developed. It is composed of six key procedures which are further decomposed into subordinate procedures. The following identification and definition of the main procedures is followed by an explanation of the SWSC implementation (or planned implementation) of that procedure.

(1) Establish Functional Project Framework

The initiation and planning of the project is performed during this activity. It includes developing business overviews, weighing and selecting objectives, determining opportunity areas, and establishing the project scope.

1. Space and Warning Systems Center (SWSC), Corporate Information Management (CIM) Team, Section 1 Introduction, *Business Process Improvement Analysis of SWSC Software Maintenance, Activity Based Costing (ABC) Foundation Workshop*, Systems Research and Applications Corporation (SRA) at Peterson AFB, Colorado Springs, CO, pg 1-1.
2. D. Appleton Company, Inc., Business Process Improvement: The CIM Initiative, *Corporate Information Management Process Improvement Methodology*, D. Appleton Company, Inc. Second Edition, (3028 Javier Road, Suite 400 Fairfax, VA 22031 (703) 573-7644 1993) pg 6.

The scope of the SWSC software maintenance process modeled and posted includes the following: activities performed by the operator/day staff, creation and coordination of Incident Reports (IRs) and Modifications (MOD) Software Change Forms (SCFs), SWSC planning baseline control, engineering and daily support, and the Integrated TW/AA Configuration Management Process.

(2) Document and Analyze Current Baseline

During this activity the current business process including the cost of each individual activity is identified. $IDEF_0$, an activity modeling technique developed by the United States Air Force, has been mandated by the CIM Information Technology Policy Board as the mandatory technique to use for modeling the business processes.

The $IDEF_0$ models are then used to drive the Activity Based Costing (ABC) process. The ABC techniques reorganize traditional financial information to show functional managers what they do with money, rather than how they spend it. It helps characterize the value of, or need for, each activity.

For the SWSC, the AS-IS process defines the software maintenance process beginning with the recognition of a problem or definition of a desired change and continues through the implementation of a vertical release. Included are the activities associated with processing SCF's through the Integrated TW/AA configuration control process.

(3) Perform Business Improvement Analysis

During this activity the team determines the applicable best business practice for improvement opportunities to the current business process and defines alternative business processes with associated cost and risk for each. Data models for the alternative processes are developed, and alternatives are evaluated for cost benefit to the baseline. The most cost-effective alternative is selected and recommended as the TO-BE business process alternative.

For the SWSC Software Maintenance Business Process Improvement Analysis, the processes being developed on the SCAI Demonstration Project are being used as the basis for creating the SWSC TO-BE activity model. It is anticipated that the SCAI process will eventually replace the current process. The actual migration of the SWSC to the SCAI megaprogramming development strategy is anticipated to require a 5 to 10 year commitment.

(4) Develop Management Plan and Functional Economic Analysis (FEA)

The activities during this process define the management decision and documentation processes. An FEA is completed on the technical alternatives and presented for management decision. The function portion of this analysis helps people understand what an organization does, how well it accomplished its mission, and how its processes can be improved by creating a picture of how the business is run on a day-to-day basis. The economic analysis helps people understand the potential value or future economic benefits of specified investments.

The FEA analysis of the SCAI process will be used to support the development of the business case which presents the benefits of using a megaprogramming development strategy. This analysis will be used to support the transition of the SCAI technologies throughout the SWSC organization.

(5) Review and Approve Program

During this step, the management decision is reviewed by appropriate approval authorities for policy, programming and acquisition. If approved, the action plan is implemented.

(6) Execute Functional Process Improvement (FPI) Program Decisions

This process puts into effect the new business plan established for the organization.

The CIM Business Process Improvement Analysis of the SWSC Software Maintenance Process and the Activity Based Costing Foundation Workshop for the high level SCAI Process Architecture has been accomplished. The workshop evaluated the SCAI approach to software development and maintenance, and appropriate alternatives to that approach from a functional and economic perspective. The following alternatives were developed by the workshop:

(1) *Alternative A* - Implement the SCAI paradigm across the SWSC organization for Space and Warning Control Systems.

(2) *Alternative B* - Using the current baseline process and augmenting it with a compilation of amicable processes and management chances to improve the existing process.

(3) *Alternative C* - The status quo. Continue to do business without making any change in the process.

A preliminary Functional Economic Analysis (FEA) was performed to evaluate the costs of alternative ways to accomplish the functional objectives of the SWSC Software Maintenance Process.

Design/IDEF, developed by Meta Software, is an automated support technology for developing IDEF models. It is being used on the SCAI to document the high level Process Architecture.

Reusable Integrated Command Center (RICC).

RICC is a technology developed for the SWSC by TRW under the Air Force Embedded Computer Resources Support Improvement Program, and includes tools for generating Ada code, definition files for applications and some reusable domain components. It was developed in response to the observation that the relatively diverse set of C^2 systems have a large subset of common requirements. The commonality or requirements suggested that significant system development and maintenance leverage could be obtained by developing reusable software to satisfy those common requirements.

Figure 1 depicts the general functions which are performed by most C^2 centers. In general C^2 centers receive mission status and data messages from external sensor sites and/or other C^2 centers. C^2 centers also output mission assessment, status, and command messages to the external sensor sites and C^2 centers. The Communication function must provide the capability to process the protocols used to communicate this message traffic. There are a number of standard protocols and consequently this function is a good candidate for incorporating into a C^2 infrastructure.

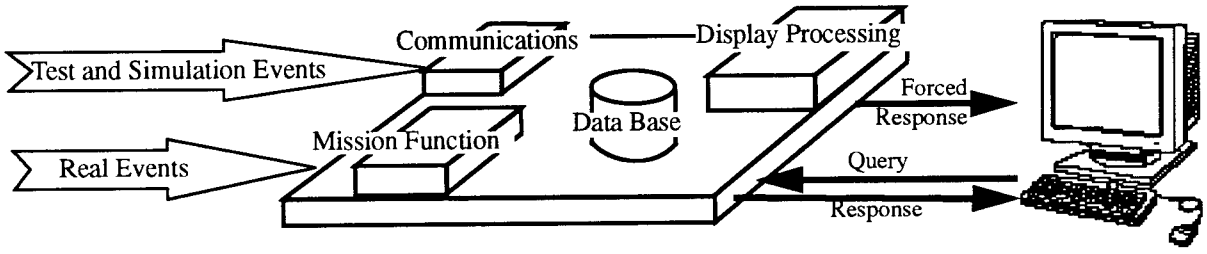


Figure 1. Typical Command Center Functionality

The Communication function provides the input and output message processing. The input message function receives inbound external messages from external entities, extracts the various fields from the message, converts the fields to internal data types as required, and validates the data for each of the fields to ensure that the message is valid. Following validation of the input message, the converted and validated data is given to the mission algorithm functions for processing or stored directly in the database. On the output side, the output message processing starts with the receipt of mission data from either the Mission Algorithms or from the Command Center User to be sent to external entities. Then the Communication

function formats the data into the correct external message format and transmits the messages. Both external input and output message formats are typically unique to the missions being performed. However, the functions performed in input and output message processing are basically common from one C^2 system to another. Thus, it is possible to develop generic data driven message processing software which is tailored for specific missions via mission specific database data describing the formats of the messages.

The Mission functions receive mission data from both external message sources and from the operator. These algorithms will process the input mission data and then make the processed data available for output to either external entities via the Communications function or to the operator via the Displays function. Mission algorithms are typically very unique to the specific mission of the C^2 center and consequently are not good candidates for a C^2 infrastructure. However, it should be noted that there are some portions of mission algorithms which are good candidates for the reuse or infrastructure. An example of this is the orbit predication function which is required by many Space and Missile related C^2 centers.

The Data Base provides the repository for information received from external sources and other information generated by the Mission functions.

The final function common to all C^2 centers is the Operator Interface. This function receives mission data from the Mission Algorithm functions, converts and formats the data as required and then outputs the data to the C^2 center operators' consoles. The data displayed occurs as a result of information forced on the user, such as alarms, or data requested by the user. This function also accepts inputs from the operator to initiate, control, and terminate mission processing. The specific displays and control features provided are normally very specific to the particular C^2 mission. However, these displays and controls can be composed from a more basic set of interface objects which are common among various C^2 centers such as menus, commands, forms, graphs, maps, etc. Thus, it is possible to develop generic, data driven operator interface software which is tailored for specific missions via mission specific database data describing the interface objects of the displays.

Significant commonality exists between C^2 systems and the common portions tend to be the most technically challenging, costly, and risky areas of the system design and implementation. By developing a reusable software infrastructure which addresses these common requirements, dramatic savings in development cost and schedule time for a given C^2 can be attained.

The C^2 Architectural Goal is to provide a reusable C^2 software infrastructure. Figure 2 is an abstract representation of the layers that are essential to the implementation of a C^2 Center, and the Command and Control Architectural Infrastructure layer represents the functionality provided by RICC and Universal Network Architecture Services (UNAS).

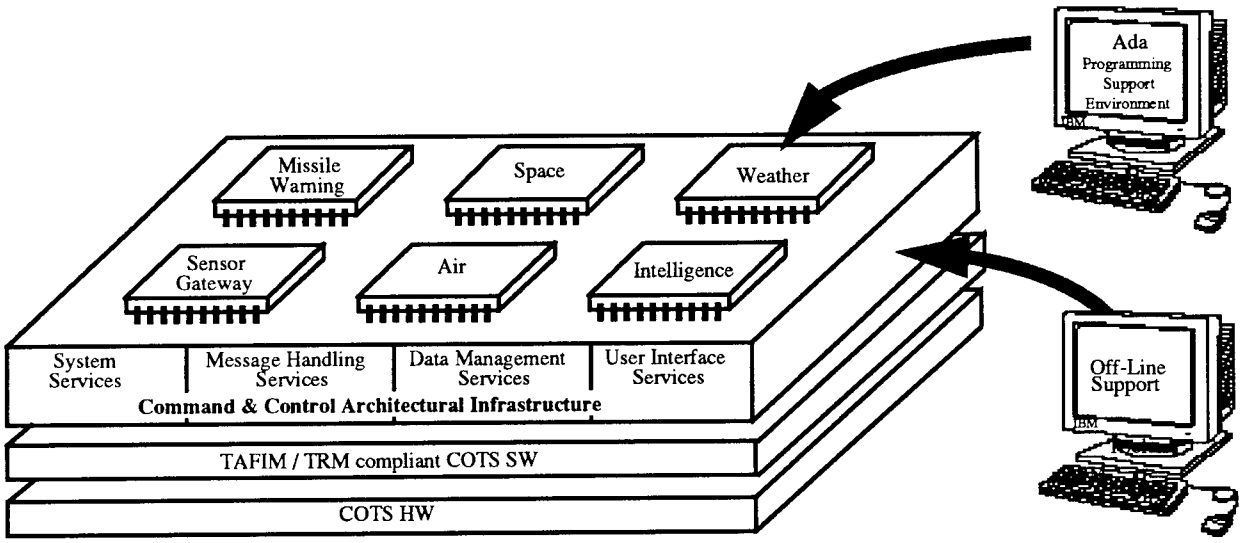


Figure 2. C^2 Architectural Goal

Mission Operation/Information Analysis (MOIA)

MOIA is an analysis method, using $IDEF_0$ notation which looks at the low-level operations of a command center from the point of view of the user. It was created to bridge the classic gap between users and developers; the transition from operational requirements to system requirements. MOIA employs a four phase methodology to analyze and define the activities and information employed by a work center in fulfilling its assigned responsibilities. Each phase is described as follows:

(1) Phase I identifies and describes the activities which are conducted in a work center. MOIA methodology is system independent and captures all activities, manual and automated. The end products of Phase I are:

- a series of MOIA diagrams using $IDEF_0$ diagraming techniques, and
- activity reports.

(2) Phase II associates external inputs with outputs, and vice versa, for each activity at its lowest level. Once the associative process is complete, then candidates for automation or improved automation are identified; allocated to existing or planned systems; and based upon the operations benefit derived from automation and frequency of occurrence, assigned a priority for Implementation.

(3) Phase III involves the analysis of existing program or cost data to determine which candidates for automation associated with a particular system are in the baseline for system development projects already underway. Candidates which are not in a baseline are identified as gaps or shortfalls.

(4) Phase IV develops a plan of action for acquisition adjustments, Pre-Planned Productivity Improvements, or other programmatic actions to satisfy all requirements. In the case of existing programs, this implementation plan recommends appropriate acquisition adjustments if any gap or shortfall was prioritized high enough to warrant changing the program baseline. For new programs, the implementation plan identifies how many of the prioritized items could be paid for with available dollars.

Each of the phases builds upon the previous phase. Once all four phases are completed on a work center, a comprehensive information flow, shortfalls, and implementation plan is available for that center.

Booch Object Oriented Analysis

Like all good object oriented methods, the Booch Method is primarily a means of developing and communicating the design for a system or a family of systems. The development of models of a system occurs in stages that allow for concentration on certain aspects of a system at a time. This approach acknowledges the distinct probability that during the first look at requirements and mapping them to design, requirements understanding will change. Therefore there is a need to continually integrate discoveries into one underlying model, that is, the process iterates between analysis and design.

The Booch Method is accomplished in three steps:

(1) Requirements Analysis provides the basic charter for the systems functions.

(2) Domain Analysis provides the key logical structure of the system.

(3) Design provides the key physical structure of the system and maps the logical structure to it.

Both analysis and design processes are iterative and incremental. When ambiguities or omissions are discovered in the analysis model it is appropriate to return to analysis activity and then continue the iterative process.

TRW Ada Process Model

Walker Royce's TRW Ada Process Model was designed to address design breakage due to early unknowns in large complex systems developments. The key strategies inherent in this approach are directly aimed at the three main contributors to software diseconomy of scale: minimizing the overhead and inaccuracy of interpersonal communications, eliminating rework, and converging requirements stability early in the development lifecycle. These objectives are achieved by:

(1) Requiring continuous and early convergence of individual solutions using Ada as the lifecycle language.

(2) Evolving solutions as rapidly as practical to eliminate ambiguities and unknowns in the problem statement by prioritizing development of real code increments of capability.

The objectives are accomplished through the following practices:

(1) Design Integration - Unlike conventional software development which enforces the concept of postponement of all coding until after Critical Design Review (CDR), the Ada Process Model requires the early development of a Software Architecture Skeleton (SAS) which corresponds directly to the top-level components and their interfaces. The SAS provides a vehicle for early interface definition by compiling these top-level components, and providing adequate drivers/stubs so that they can be executed to evaluate design quality. This forces early baselining of the software interfaces which in turn permits smooth development and avoidance of downstream breakage.

(2) Demonstration-Based Design Review - Again, unlike conventional approaches which rely on paper demonstrations of design correctness, the Ada Process Model employs demonstrations during design reviews to provide validation of design correctness.

(3) Total Quality Management -The use of Ada throughout the lifecycle permits consistent software metrics across the software development work force and the demonstrations serve to provide software developers with tangible progress indicators.

(4) Incremental Development - This well-known software engineering technique is employed because of the need to adjust build content and schedule as more accurate assessments of all factors can be made.

STARS Technologies

STARS Process-Driven Development

To improve software quality and productivity, STARS felt it was necessary to focus attention on the software processes being employed and the means used for delivering them. The success of software development organizations is directly related to the quality of the products they produce. When organizations have no training or consistent approach to repeat their results, they are dependent on the resourcefulness of the people they employ and they are likely to achieve pockets of success rather than success across all levels of the organization.

The concepts and approaches which define the STARS Process Driven Development approach for increasing the productivity, reliability and quality of critical government software systems are designed to help organizations and projects achieve process capabilities such as those characterized by the higher levels of practice in the Software Engineering Institute's (SEI's) Capability Maturity Model (CMM).[1]

What Process Driven Development means to organizations is that:[2]

1. Technical Report 85.0170, STARS Process Concepts Summary, December 1992, Hal Hart (TRW), Jerry Doland (Paramax), Dick Drake (IBM), William Ett (IBM), Jim King (Boeing), Herb Krasner (Krasner Consulting), Leon J. Osterweil (Univ. of California at Irvine), James Over (SEI), Terri Payton (Paramax), Coordinated by Learning Resource Center, IBM, Gaithersburg, Maryland 20879 pg 2.
2. Technical Report 85.0170, STARS Process Concepts Summary, pg 2.

(1) Organizational processes are formally or semi-formally defined, are globally known and visible throughout the organization, and are adaptable and tailorable to meet project and product goals.

(2) System and software development is guided by a defined process for all activities from the system inception (or beginning of the organization's involvement) through system deployment, evolution, and eventual retirement.

(3) Environments and tools are integrated to support a defined process.

(4) Defined processes promote collaboration and teamwork by making activities, roles, and dependencies clear.

(5) Process definitions are developed, maintained, evolved, and reused with a level of concern and discipline approaching that applied to software products themselves.

(6) Process definition disciplines are improved through manual and automated measurement and feedback techniques.

(7) Process definitions are installed and guidance is available through documentation or tied to automated tool invocation.

The early efforts of the STARS program were to explore innovative solutions to improve the management and control of the complex and varied activities (software process management) required to develop, field, and support Mission Critical Computer Resource (MCCR) software. STARS aimed to establish capabilities to support tailorable process definition and management which involved the ability to define, monitor, measure, control, and continuously improve the activities that make up the software lifecycle processes.[1] Software process management is a key focus of the STARS process-driven paradigm, and the Loral STARS team has focused attention on providing automated support for process-driven project planning, measurements, and automated process enactment to further an organization's ability to manage and control a project and its processes.

Several technologies have been developed to support the STARS process definition and improvement strategy. Process definition requires the specification of the identified processes, where specification includes a process model and a process definition document (also referred to as a process guide) which describes the relevant set of activities, artifacts, and agents; the relationships within and among those three classes; and the behavior of the entire set of entities and relationships. Given these documents, the process can be enacted by humans. [2] Processes may be defined so that they are manually enactable, where a human follows a written procedure, or automated, where a computer coaches a human on what to do next given their current state. The focus on the SCAI project is to support both manual and automated enactment.

Automated Process Support Tools

The enactment of SCAI processes is supported by a number of automated tools, namely: ProjectCatalyst, Process Weaver and SPMS. CAT Compass and Amadeus support project and process management.

ProjectCatalyst

ProjectCatalyst was developed by Loral STARS teammate Software Engineering Technology, Incorporated (SET) and is a family of integrated software components which:

(1) Supports task dispatching and monitoring.

(2) Guides the engineer in following the defined process.

1. STARS Workshop Process Management, Dick Drake, STARS Newsletter, Volume II, Number 1, March 1991 pg 5.
2. James W. Armitage, Marc I. Kellner, Richard W. Phillips, Software Process Definition Guide, Content of Enactable Software Process Definitions, Guide SEI-93-SR-18, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, August 1993, pages 2-6.

(3) Provides the necessary files and tools to the engineer performing process tasks.

(4) Manages complex parallel process activities.

(5) Facilitates team communication by automatically maintaining task status and tracking task prerequisites.

(6) Reports task, product and milestone status to SPMS.

(7) Gathers effort, schedule and quality measures for presentation by Amadeus.

ProjectCatalyst provides these functions through the use of Process Weaver.

Process Weaver

Process Weaver was developed by Cap Gemini. It functions as the process enactment component of the ProjectCatalyst tools. It provides the interface to the users and supports the management of dispatched tasks.

Software Process Management System

The Software Process Management System was developed with Loral STARS teammate Cedar Creek Process Engineering and supports process definition through process modeling. SPMS supports an Entry, Task, Validation, Exit (ETVX) process definition paradigm, which is conducive to the enactable process definition focus of Information Organization Templates.

SPMS provides the capabilities to model and continually refine an organization's process definition. It also provides the capabilities to instantiate the defined process for a particular project through the inclusion of time and resource scheduling. SPMS interfaces with a project planner package, CAT Compass, to provide project management with process-driven project management capabilities. SPMS includes support for:

(1) Process and measurement definition

(2) Process driven plan generation

(3) Scheduling

(4) Resource allocation

(5) Plan simulation

(6) Plan monitoring

SPMS will allow the SCAI project to focus on the vital process issues of cost, schedule, and quality.

CAT Compass

Cat Compass provides the project planning and reporting capability.

Amadeus

Amadeus was developed by Amadeus Software Research and is a software product that automates measurement activities within a software engineering environment. Amadeus consists of several software components, the heart of which is an interpreter. This component monitors significant events such as tool invocations, changes in source files or documents, creation of software problem reports, expiration of time intervals, or changes in collected measurements, and invokes other components called agents, to respond to the events. Some of the agents perform collection of measurement data, while others analyze the data or produce visualizations of the data, or feed it back into the development process.

Cleanroom Software Engineering Process

The Cleanroom Software Engineering approach to quality software development is based on over 25 years of development, verification and testing experience by Dr. Harlan D. Mills and his associates, first at IBM

and subsequently at SET. Cleanroom provides a tightly integrated approach from specification preparation through software development to software certification, which brings engineering rigor and intellectual control to the software development process. Cleanroom processes and practices have demonstrated improved programmer productivity and software correctness and reliability.

The fundamental notion of Cleanroom is the necessity for the engineering staff to maintain intellectual control over the project. Intellectual control is the ability to clearly understand and describe the problem at hand at the desired level of abstraction. Cleanroom uses a number of organizational and technological strategies to help the engineer maintain intellectual control over the software project.

With Cleanroom, there are three teams that each handle different aspects of the software development process. The teams are as follows:

(1) The Specification Team prepares and maintains the specifications.

(2) The Development Team designs and builds one or more software increments. The resulting source code, prior to any compilation, is turned over to the Certification Team. They are also responsible for isolating and making any changes necessary to the increment as a result of problems. Large projects have multiple Development Teams.

(3) Each Certification Team prepares test cases for an increment and when the increment is submitted for certification they perform the tests and prepare the certification report. The Certification Team executes the code, but does not modify it in any way.

Cleanroom specifications are developed in a spiral manner, with each iteration making the specifications more complete. A Cleanroom Specification is comprised of six volumes:

(1) Mission: A precise statement of the requirements for the software system, primarily functionality and performance.

(2) User's Reference Manual: A definition of the stimuli and responses invented so the software can fulfill its mission. This volume addresses 'look and feel' issues, as well as performance.

(3) Software Function: A black box function that defines software responses in terms of stimuli histories (i.e., in an implementation-free manner). This volume clearly describes the functional behavior of the software.

(4) Specification Verification: A rigorous argument that the software as defined will meet its defined mission.

(5) Software Usage Profile: A Markov Model stating the probability of moving from each usage state to all other usage states. This volume describes how the software is to be used, in terms of sequences of state transitions, according to a projection of how the software will actually be used. Every possible state that a system can be in is described, with probabilities associated with all possible next states that can be reached from a particular state as a result of a single transition. The Usage Profile can be represented in terms of a state transition diagram or a matrix. This volume may also include test fragments for each transition, which can be concatenated, as a result of statistically generating a path through the software, into a test case.

(6) Construction Plan: A plan for building the software in a series of increments such that each accumulation of increments is executable by user stimuli. The development of the construction plan requires significant effort to determine a definable series of user executed increments. Increments of 5,000 to 10,000 lines of code, which require approximately 6 to 8 weeks of effort for a Development Team, typically have been utilized.

Domain Analysis Process Model

Reuben Prieto-Diaz created a systematic, repeatable process for doing Domain Analysis, which can be described as: "systems analysis for a class of systems", or "the activity of identifying the objects and operations of a class of similar systems in particular problem domain". The objective of Domain Analysis is to discover and define domain models and architectures common to a family of applications for supporting pre-planned reuse.

Following is a sketch of the Domain Analysis Process at a very high level:

(1) Select the domain with the highest reuse potential

- Look at current projects: scope and define the domain
- Evaluate current/future needs, current practice, feasibility
- Define purpose

(2) Top-Down Analysis

- Identify high level architecture and functional model
- Select functional components with high reuse potential
- Re-define architecture (with reuse in mind)

(3) Bottom-up analysis

- Vocabulary analysis
- Classification model
- Functional clustering

(4) Derive generic architecture

- Map bottom-up functions into architecture
- Adapt architecture
- Derive other models

Outputs of this process are:

(1) Domain definition

(2) High level domain model/architecture

(3) Faceted classification

(4) Domain vocabulary

(5) Reusable structures

The unique aspect of the process is the bottom-up part of the analysis. The contention is that by examining systems and systems documentation, common terms can be derived that are general to the whole domain. More specific terms can be grouped under more general terms called facets. The more specific terms are deemed facet-terms. So, potentially, a domain analyst can uncover generality simply by understanding language. Relationships can be generated between the facets so that "standard descriptors" can be generated for the domain. Standard descriptors can represent generic functions in the domain. For example, an

early attempt at reuse was when Booch defined a set of general Ada Components managing standard data types like stacks and queues. A standard descriptor for Booch Components, where parenthesized items are facets, would be: Component of type (Role) consists of or operates on (Structure), has (Concurrency) and (Space_form), performs (functions(s)) on (object(s)), using (Method).

SCAI/STARS/SEI Process-Driven Technologies

Information Organizer Templates

Information Organizer Templates, a mechanism to support the definition of manually enactable processes, has been developed as a result of a Process Definition Technology Partnership between the SCAI project and the Software Engineering Institute (SEI). The partnership was intended to develop prototype products, and collaborate on solutions to technical problems. The templates identify all the information required to make a process manually enactable. In order to avoid overwhelming the potential process definers or process users, the team devised a way to subdivide the information into categories, which also represent convenient stages for the process definer to follow when developing the process definition.[1]

(1) Stage 1: **Process Layout:** Show overall flow of activities and work products defining what is to be accomplished by a given process; available in an enactable process guide for reference as a navigation aid during enactment.

(2) Stage 2: **Process Design:** Include lower level refinements beyond information necessary to just depict process layout, such as the methods that describe how an activity's results are to be accomplished, and the agents responsible for performing the activities. This information is documented in an enactable process guide for reference as needed prior to and during enactment.

(3) Stage 3: **Process Enactment:** Include the process information that is necessary during enactment to govern the process (such as activity and artifact states) or information that describes specific process discipline actions to perform during enactment (such as when to verify or validate a result, when to record product and process metrics, when to log status, when to communicate status to others, or solicit status from others, and how to determine which activity in the process is to be performed next upon completion of a given activity).

(4) Define a set of metrics which could verify the successful achievement of the goals.

(5) Establish an approach for capture and collection of the metrics.

(6) Define an approach to presenting and analyzing the measurement results.

(7) Institute a continuous improvement cycle for the measurement process itself.

The approach to metrics definition will be both top down and bottom up. From a top down perspective the project goals will be defined and refined. The purpose of the metrics activity is to determine if the goals are being reached. The goals provide the rationale for the actual measurements collected. You could also think of the goals as providing requirements on the metrics activity.

1. Linda Parker Gates, Richard W. Phillips, STARS/SEI Technology Transition Experience Report November 30, 1993, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania pg.19

Appendix C

Megaprogramming: Enabling the Future SWSC Product-Line

The SWSC is responsible for the maintenance of a wide range of space and warning C^2 applications. The SWSC's long-range intent is to manage as many systems as possible within this domain of applications using a coherent process; and the envisioned product-line organization is intended to facilitate this type of management. There are three fundamental prerequisites for a successful product-line organization: a domain manager responsible for all applications in the product-line, sufficient commonality among the applications to allow management of the entire set as a whole, and enabling technology to facilitate the systematic management and engineering work.

Domain Manager

Figure 1 depicts a possible future product-line organization and shows how the three STARS technology areas combine to enable a product-line way of doing business. The SWSC Commander would serve as the Domain Manager in this organization. Ideally, the Domain Manager will be in a position to make trade-offs between individual application objectives and overall product-line evolution objectives. As an example of this type of trade-off, it might be appropriate to build a new reusable asset that will greatly enhance the efficiency of the total organization (enabling the SWSC to build future systems "cheaper, better, faster"), but in order to build this new asset, there might be some impact to the cost and schedule of one or more applications.

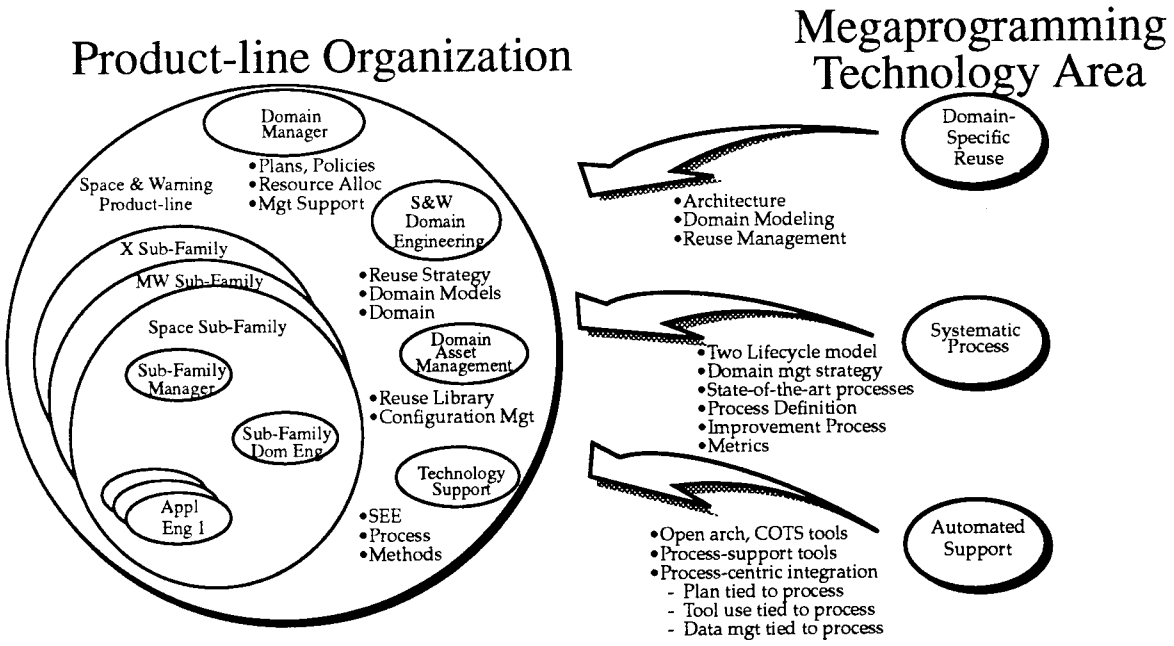


Figure 1. Megaprogramming: Enabling a Future Product-line Organization

Application Sub-families

Applications in the SWSC product-line will be developed and maintained using a common Application Engineering process that maximizes the reuse of common product-line assets, including process compo-

nents, software components, models, software engineering environments and tools, etc. As shown in Figure 1, these applications may be further divided into sub-families based on the opportunity for additional layers of reuse.

Domain Engineering

Domain Engineering is the engineering arm of the organization charged with determining the commonality in the requirements of the domain's applications and deriving and evolving the best architectural approach to building those applications - which includes identifying reusable software components, code generators, and the like.

Domain Asset Management

Domain Asset Management provides storage and management of all product-line assets of all types (process components, software components, models, specifications, etc.), facilitating the proper use of those assets across the entire product-line.

Technology Support

Technology Support identifies, develops, evolves and supports the organization's common processes and methods. It also provides the software engineering environment, which provides automated assistance for applying the processes.

Megaprogramming Technology Emphases During the Demonstration Project

To prepare for this type of organization, the SWSC is working with STARS to mature its technological approach, using the SCAI Demonstration Project as its primary vehicle. As discussed below, the SWSC and its existing contractor base have already made substantial progress towards megaprogramming; thus, more attention is being given to some areas than others.

(1) **Domain-specific Reuse**

Prior to forming the partnership with STARS, the SWSC had already established an excellent head-start in this technology area through its work with TRW on a reusable architectural infrastructure - culminating in the Reusable Integrated Command Center (RICC). This provided an architectural starting point for the SCAI. The SWSC's primary focus in domain-specific reuse during the SCAI project is systematizing the use of the architectural infrastructure and building up Domain and Application Engineering (DE and AE) processes. Heavy emphasis is being given to various types of modeling as a means of capturing the requirements for the SCAI application, specifying its architecture, and following these models through to the executing system. The abstractions being developed during this activity are being chosen to insure that the models will form the starting point for the product-line's Domain Engineering modeling work.

(2) **Systematic Process**

This technology area is the one that is receiving the most emphasis by the Air Force/STARS partnership. The intent is to build up a coherent process for the SCAI, with emphasis on Application Engineering, and to establish a repeatable product-line method for recording, modeling, managing, and applying the process.

(3) **Automated Support**

STARS is working with the SWSC to build up an integrated software engineering environment (SEE) to support the SCAI process. Notably, STARS is supplying two new process support tools and is supporting the integration of those tools with the rest of the SEE to provide automated support for the process.

Appendix D

Iterative Technology Assimilation and Evolution

Applicability of the STARS CFRP

One of the key principles espoused by STARS is the Plan/Enact/Learn paradigm, as detailed in the STARS Conceptual Framework for Reuse Processes (CFRP), as illustrated in Figure 1.

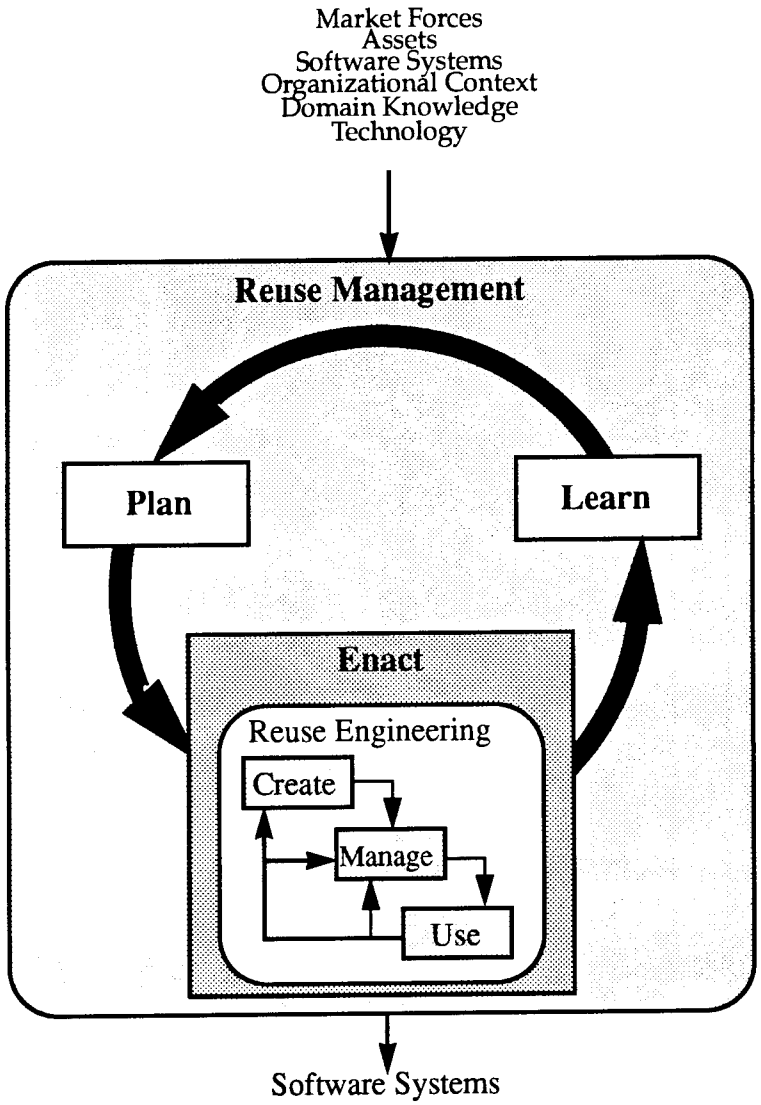


Figure 1. Conceptual Framework for Reuse Processes (CFRP)

The essence of this framework is that introducing anything significantly new is best viewed as an iterative undertaking; and that if the intent is to reuse the results, the iteration must be consciously managed. Thus, incremental changes are planned, the changes are applied, and the experience gained in applying the changes is used to adjust the plan for the next iteration.

For the SCAI project, it is meaningful to consider the CFRP from various points of view, notably:

(1) Formulating an overall technical approach,

(2) Formulating specific approaches in each of the four experience areas,

(3) Developing reusable assets for the domain, and

(4) Developing the SCAI application, including reuse of domain assets.

The first two deal with technology transition and process definition, which were the main activities addressed by the team during the Preparation Phase. The last two deal with producing application and domain products. At first glance, the CFRP seems to pertain primarily to items 3 and 4, but the SCAI team's experience has demonstrated that it in fact pertains equally well to items 1 and 2. If this fact had been better appreciated at the outset, the team might have been able to achieve consensus on the approach more quickly.

In general, the team has found that an iterative approach permeates nearly all significant technical activities.

The following paragraphs briefly discuss the Preparation Phase experience from each of the above points of view, in the light of the CFRP:

(1) Formulating an overall technical approach:

Several characteristics were identified by the SMX directorate that were considered essential to the approach. It must:

- allow for early buy-in by the customer,
- allow for iterative development of the system,
- support product-line development, and
- be evolutionarily in nature.

The CFRP iteration began with the Learn point of the cycle with a joint review of the SCAI objectives and a review of the existing SWSC and STARS technologies and progressed into a cycle of synthesizing candidate solutions, discussing and documenting them, assessing their viability, and moving on to the next iteration. Each successive iteration would typically involve refining/adjusting the solutions defined in the prior iteration, and integrating new aspects of the approach not previously considered.

At the outset, with the large amount of learning needed on all sides, it was very hard to develop specific plans for developing the approach. Given the number and sophistication of the technological changes contemplated, a large number of iterations were needed before the approach really began to gel. In fact, the first attempt to capture the total approach in a coherent description was in Version 2.0 of the SCAI Demonstration Project Management Plan (SDPMP), published at the end of the Preparation period (10/15/93). Although SCAI development began shortly thereafter, the project realizes that a significant amount of iteration remains ahead - and in fact, the iteration really continues throughout the life of the product-line.

The following discussion provides more detail about how this iteration has progressed to date.

The multi-organizational project had to understand the variety of SWSC incumbent technologies, the organizations in which those technologies were used, and the STARS technologies, before the participants could begin the process of integrating the technologies into the SCAI technical approach. The approach needed to integrate the STARS technologies with the SCAI incumbent technologies such that the resulting approach met the characteristics identified by the SMX directorate. The steps that the project planned to iterate through to develop the approach were:

- Understand the organizations contributing to the SCAI project.
- Decide/Plan how to integrate incumbent technologies with STARS technologies.
- Test the merged technologies in Pilot Projects.
- Record the processes for applying these merged technologies.
- Learn lessons about the merged technologies and record the lessons.
- Incrementally, integrate the tool-set associated with the merged technologies.
- Enact selected merged processes

(2) Formulating specific approaches in each of the following experience areas:

The CFRP applies equally well to the activities the project has followed in developing the specific approaches and instrumentation of those approaches in each of the experience areas discussed in this report. In brief, the project has experienced the following:

- **Domain/Application Engineering**: Defining the DE/AE approach during the Preparation Phase involved acquiring both domain and technology knowledge (learn), defining/refining the candidate approach and assessing its viability (enact), and determining which aspects to address in the next iteration (plan). At each iteration, the team matured its understanding of both the domain and the technology. Further, at each iteration, the team added more depth to the approach. This experience is recounted in detail in the DE/AE section.

  Another aspect in which the CFRP applies is to the resulting DE/AE approach itself. Domain Engineering for the SWSC is viewed as an iterative activity, starting with the Application Engineering for the SCAI and using the SCAI AE products as the start of the DE iteration. As new applications are built, the new experience gained is fed back into the DE products.

- **Process Support**: The project has iterated on its method for defining a process. As the overall project approach matured, several iterations of the process architecture occurred. At each stage, the team learned more about how the $IDEF_0$ process modeling method should be applied and how it should be combined with other modeling methods. At the end of the Preparation Phase, a plan was put in place to use the adopted process modeling approach to build up the SCAI process incrementally during the Performance Phase - starting with the Specification Process. The modeling, and later the enactment, will be done with automated assistance from a STARS-sponsored process tool suite. It is expected that both the process and the tool suite will be revised incrementally as the project gains experience with their use.

- **SEE Support**: SEE assembly and integration is being performed iteratively as well, especially in view of the iterative development of the process.

- **Metrics**: The metrics iteration started with the definition and documentation of the SCAI project goals. As the project proceeded, the goals were iterated to reflect

experience. In addition, the metrics instrumentation in the SEE is being built up iteratively.

(3) Developing reusable assets for the domain

The SWSC has already done a great deal of iteration to build up its architectural infrastructure capability. The RICC tool set, which is an outgrowth of prior production work, has gone through two pilot activities to assess its applicability to the SWSC domain. At each stage of the iteration, the experience gained was used to enhance the tool set and the methods used to apply it. Further refinements are anticipated as the SCAI Application Engineering proceeds through its planned incremental development cycle.

(4) Developing the SCAI application, including reuse of domain assets.

During the Preparation Phase, the team progressed through several iterations of its domain and application modeling work. At each stage, the team enhanced its domain knowledge and increased the depth of the models as well. At the close of the Preparation Phase, the team had begun its specification work, using these models as a basis. During the Performance Phase, three SCAI increments are planned. At each stage, experience gained will be used to update and refine both the models and the specifications.

Appendix E

Preliminary DE/AE Approach

This Appendix provides additional details on the SCAI team's DE/AE experience during the Preparation Phase. This information will provide insight into the problems faced by the SCAI team while they worked to develop their approach, as well as the preliminary solutions to some of the problems. The material should be of interest to practitioners in this technology area.

It is important to recognize the preliminary nature of this information. At the conclusion of the Preparation Phase, the team had established an overall DE/AE approach, had defined a high-level project process architecture, and had initiated an iterative cycle of formal process definition. Accordingly, much of this material is only partially developed. During the next experience interval, the DE/AE process definition will continue to mature - with the help of early enactment of selected process components on Releases 1 and 2 of the SCAI application.

This appendix includes the following major sections:

(1) Selected DE/AE Approach Topics

(2) Creating a DE/AE Process

(3) Preliminary SCAI DE/AE Process

Selected DE/AE Approach Topics

This subsection discusses several topics that have proven especially important to the team in guiding the approach definition work during the Preparation Phase.

Architectural Infrastructure Chip Model

Figure 1 illustrates a typical Command and Control (C^2) system of the type maintained by the SWSC.

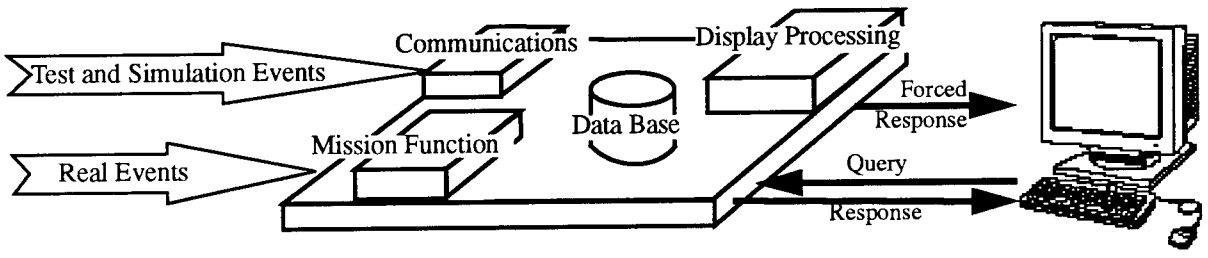


Figure 1. Typical Command Center Functionality

The SWSC has been attempting to determine a common architectural strategy for such systems for several years, culminating in the RICC architectural infrastructure approach, which is detailed in Appendix B.

The so-called "Chip Diagram", shown as Figure 2, represents the current architecture concept. The Demonstration Project is basing the SCAI application on this infrastructure, which is enabled by the RICC tools.

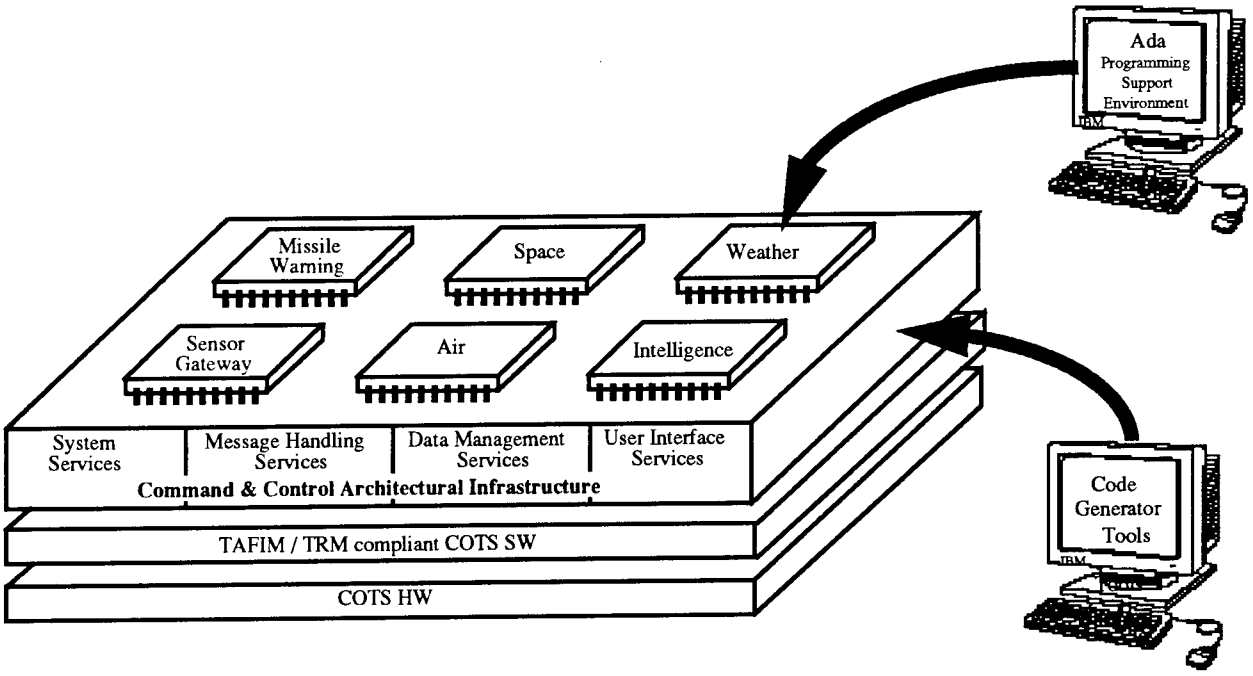


Figure 2. C^2 Architectural Goal

Layered Model Framework

The RICC two layer C^2AI chip model, shown above, was abstracted into the layering scheme, which is called the Layered Model Framework, by adding extra model layers to identify types of reusable components other than the common service routines, and to further insulate the domain components from change.

The RICC Infrastructure Services, identified as Common Services in Figure 3, are services which are applicable to the whole C^2AI domain. As new systems are analyzed and reengineered via the Domain Engineering /Application Engineering (DE/AE) process, the intention is to identify multiple instances of existing application services, and generalize them into new Common Services. Common Services act as Servers to Requesters in layers above the Common Services Layer.

An Algorithmic Layer, containing common mathematical services, potentially applicable to all SWSC systems, has been defined. These Algorithms, such as for Orbit Determination, are applicable to more than one system in the SWSC domain, but probably not applicable to the entire C^2AI domain.

Both the Common Services and Algorithmic Layers act as Servers to the Mission (or Event Layer). The Mission layer models events the system must respond to. It is presumed that there are few missions that are general to all systems in the domain. Therefore, most Missions reside in the Mission Layer of an Application Architecture Model, not the Domain Architecture Model.

A Domain Requirements Model (DRM) is a logical, processor-independent, representation of the common objects in all systems in the domain. Common domain requirements are maintained in the Cleanroom Six Volume System Specification for the domain.

If the layering structure and rules for using the layers in the domain are consistent across that domain, then the DRM can be built iteratively. The characteristic layering structure chosen for our Domain Models should be applicable to the whole SWSC domain and the Space Domain Model Common Services Layer should remain relatively free from change, even as the scope of the Domain Analysis is extended beyond the Space Subdomain.

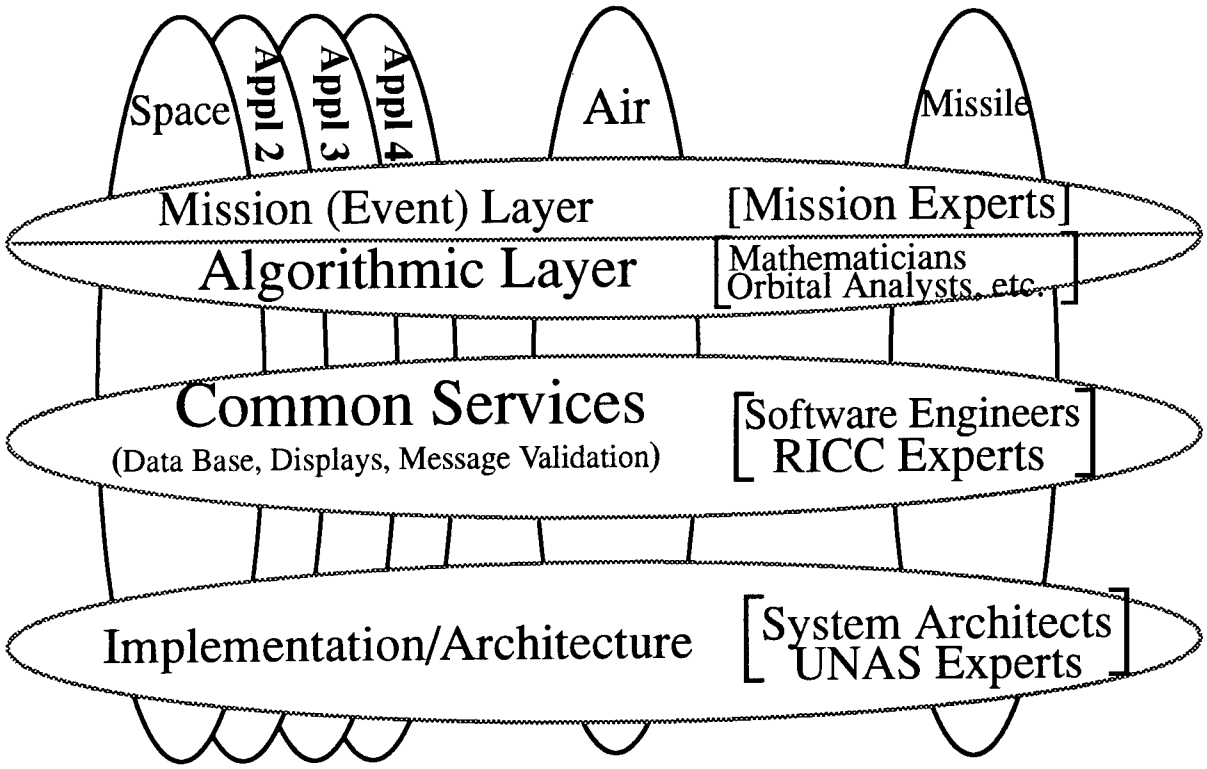


Figure 3. Layered Model Framework

The anticipated value of this approach is threefold:

(1) The relatively cheap a-priori identification of dominant domain characteristic is substituted for the expensive a-priori Domain Engineering and asset creation for the entire domain.

(2) Domain Models can be validated by use on single systems. Errors are caught before they become expensive. This is consistent with modern Software Engineering Principles.

(3) Other organizations can examine SCAI processes and technologies after they are validated, and accurately predict the cost of making the same technology transition, without factoring in the cost of a wide-scoped Domain Engineering.

The DRM layering scheme not only helps identify reuse opportunities, but also identifies the type expertise that can be localized in product-line functional organizations.

Experts who understand the Missions in the SWSC domain need only understand the Mission Layer of the Domain Requirements Model, and need not be software engineers or system architects. These mission domain experts can collaborate to identify and abstract common missions across systems; they would benefit from having worked on the teams that did the CIM modeling and MOIA predecessor work.

Experts who understand orbits, trajectories and the mathematics behind these topics, can be shared across appropriate SWSC supported missions, as part of an organization supporting the Algorithmic Layer.

Experts who understand typical software engineering tasks like relational data bases, or message parsing, can be considered as a pool available to all SWSC missions. They conceptually belong to the Service Layer of the SWSC, and should be responsible for upgrading and extending RICC services, or substituting services equivalent to RICC.

Architects, experts in creating UNAS System Architecture Skeletons and running trade studies to evaluate system performance, can be viewed as a pool of experience across the SWSC. These experts should also

understand how to identify common templates that enforce common structure or mapping rules when converting the DRM into the Domain Architectural Model (DAM). These experts support the implementation/architectural layer of the DRM, which is a conceptual layer only, and does not relate to actual Ada components, as do the other DRM layers.

Also being advocated, was a Domain Architecture Model, constructed using the Shlaer-Mellor Recursive Design Technique. The architecture model structure, though not the modeling process, was selected by the SCAI project. This structure was consistent with the architectural model structure advocated by Sholom Cohen and his Feature Oriented Domain Analysis.

SCAI Relationship Between Domain and Application Models

Domain and Application Requirements Models (DRM and ARM) are viewed as a single Booch Class Model. Referring to Figure 4, classes (and resultant Ada Packages) are either specific to a single application or general to all applications. Differences between applications are captured using child classes. Similarities between systems are captured using parent classes. Object Scenario diagrams in the Event Layer are identified as either common to the whole domain or specific to a single system. For example, the Manual Orbit Determination Scenario is common to Systems A, B, and C. It uses an Observation object which is also common to all three systems. However, Manual Orbit Determination also uses a Sensor object, which has different children for Systems B and C.

SCAI Relationship Between Requirements and Architecture Models

Referring again to Figure 4, there is a mapping shown between the requirements model and the application code of two applications in the domain. This mapping is accomplished in accordance with the "prescriptive" portion of the Domain Architecture Model (DAM). This portion provides rules for transforming the requirements model into application software. Theoretically, if the requirements model is complete enough, and if the architectural mapping rules are rigorous enough, this mapping could be automated - i.e., the application could be built from the requirements model alone. While this is not a current focus of the SCAI team, interested readers are referred to the Shlaer-Mellor Recursive Design methodology, which claims to enable such automation.

Not shown explicitly on this diagram is the "descriptive" aspect of the DAM, which is intended to describe the system to human analysts attempting to gain intellectual control over how applications are constructed.

The SCAI team views the descriptive portion of the DAM to overlap heavily with the DRM. As can be seen from Figure 4, some aspects of the architecture of the applications are visible from the OO model. Ideally, one will be able to easily trace between the as-built application software and the requirements model which led to it.

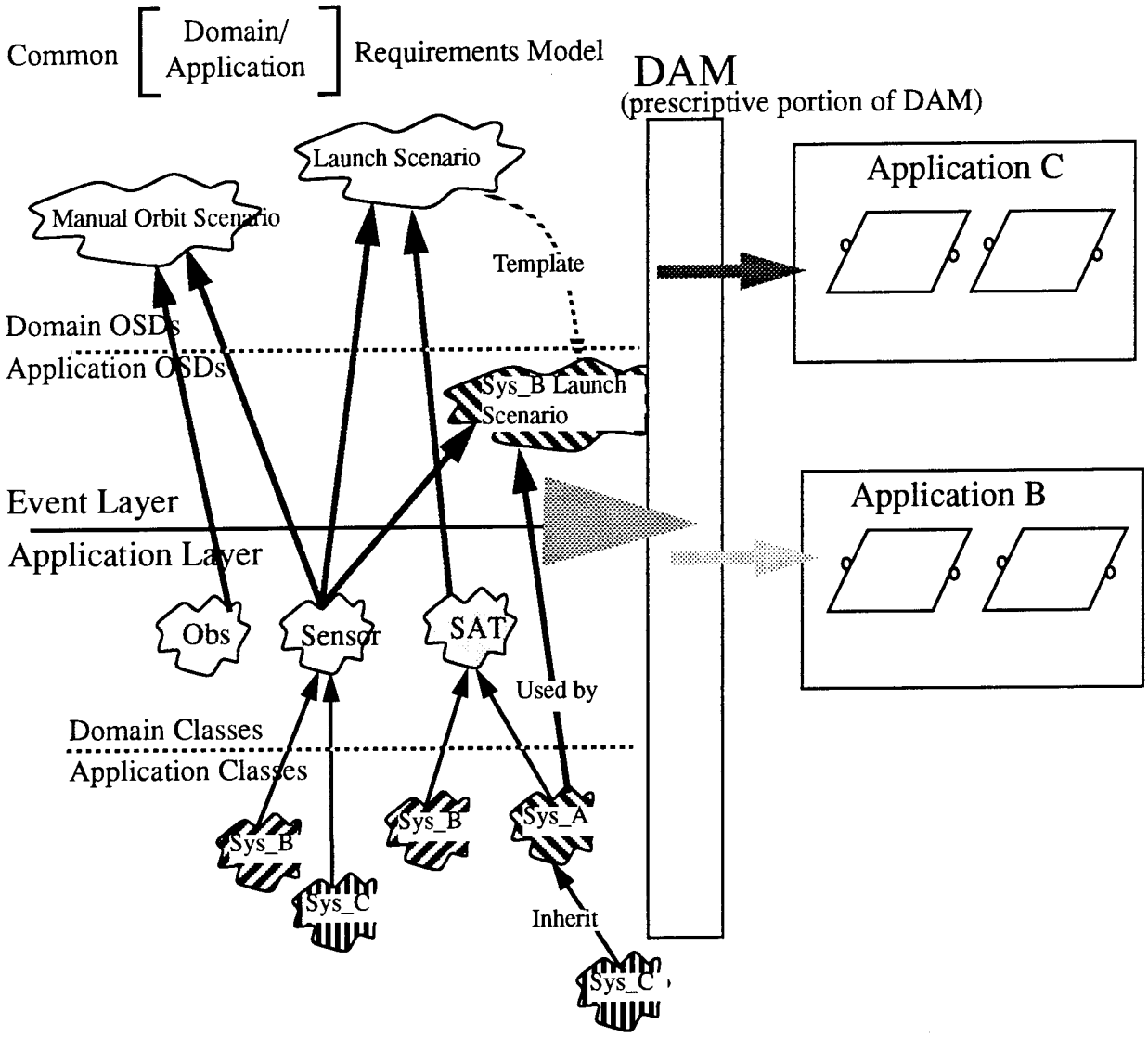


Figure 4. Domain/Application Model Relationship

Creating a DE/AE Process

Based on the experience in iterating on the DE/AE approach, the team has established a working model of how to go about defining the DE/AE process (i.e., a formal, well-defined process embodiment of the approach). It should be pointed out that this amounts to a hypothesis, since the team was still at the early stages of formal process definition at the conclusion of the Preparation Phase.

The following is an outline of the approach to defining the DE/AE Process (Iterate through the following steps):

(1) Develop/Refine and Document the High-Level Approach - including the overall project process architecture (showing how the DE/AE process works in context with the rest of the project process).

(2) Define/Refine the Process Activity Flow: this describes the work flow of the major process activities, including the identification of the key artifacts being communicated among the activities.

(3) Define/Refine the Engineering Artifacts: this describes each of the key artifacts mentioned in the Activity Flow (such as the Cleanroom 6-vol Specification, Object Model, and Architecture Model) within the context of an overall framework; at this point, the layering notions associated with the RICC architectural infrastructure are introduced, since they pervade all the artifacts.

(4) Define/Refine the Mapping between the Artifacts: this specifies how the artifacts tie together into a cohesive whole.

(5) Select/Adjust an Architectural Framework.

(6) Detail /Improve the Process using Information Organizer Templates: this captures the process to the level needed for practitioners who are using it to build product-line applications. Rather than attempt to define the entire process at once, an incremental approach is followed. Modest portions are chosen for detailing, and the experience gained in executing the selected portions is used to guide further definition activities.

(7) Use the Process and Provide Feedback to the process definition activity to help adjust the approach and improve the process.

Preliminary DE/AE Process

The prior section presented an iterative high-level "process for defining an organization's DE/AE process". This section traces this meta-process for the SCAI DE/AE process, illustrating each step with considerations and examples taken from SCAI experience.

This is clearly not a complete process yet. Further elaboration will continue throughout the project. One of the main sources of practical input that will help mature these notions is the ongoing SCAI engineering work.

Develop/Refine and Document the High-Level Approach

(1) Two Lifecycle Model - as interpreted for the SCAI

Figure 5, a version of the STARS Two Lifecycle Process Model, shows the Domain Engineering Process occurring in parallel with the Application Engineering process and suggest that existing systems be analyzed, via a Domain Engineering Process, to identify reusable structures prior to the initiation of Application Engineering. Reusable components are made available before the start of systems implementation.

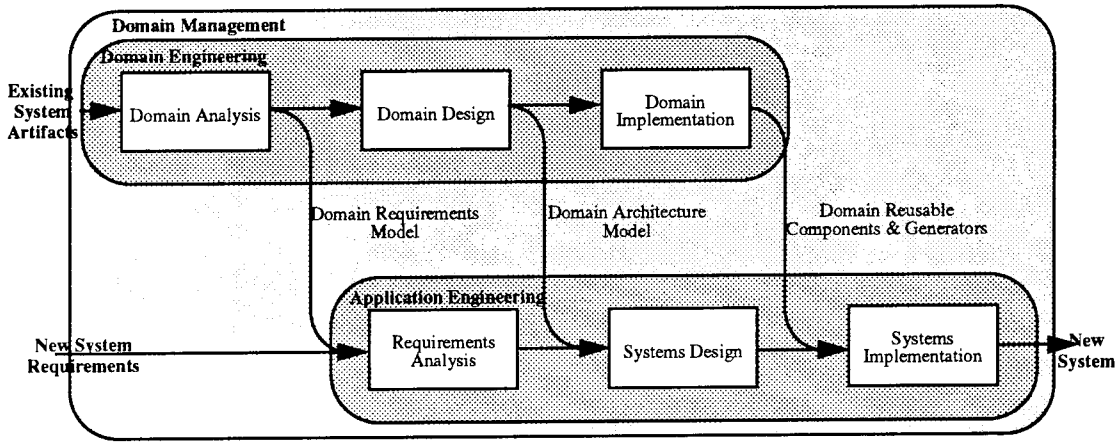


Figure 5. Two Lifecycle Process Model

For the SWSC domain, the SCAI Team postulated that given the existence of the RICC technologies, and the extensive experience the space and warning domain experts had in analyzing systems in the C^2AI domain, an informal domain analysis for the SCAI C^2 domain had already been done. The MOIA and CIM modeling efforts were also providing space mission operational models as input to the Space domain modeling effort. It was determined, therefore, that the initial project efforts should emphasize the development of Application Engineering processes, and artifacts.

The approach taken to verify that the RICC C^2 architectural infrastructure could be applied to the space domain was to perform a pilot in which two representative space mission capabilities (threads) were chosen for implementation using the infrastructure. These were Satellite Ground Tracks and Sensor-Satellite Look Angles.

The SCAI approach was to develop a model of the space portion of the C^2 domain because the entire C^2 was too large. The team developing the Space Domain Model acknowledged the RICC as the SCAI architectural infrastructure, and took an incremental approach to developing both models and modeling methodology.

As a result of the informal domain analysis that had been performed prior to the beginning of the SCAI project, the DE team adapted the STARS Two Lifecycle Process to reflect the SCAI approach to building modeling artifacts. (See Figure 6.)

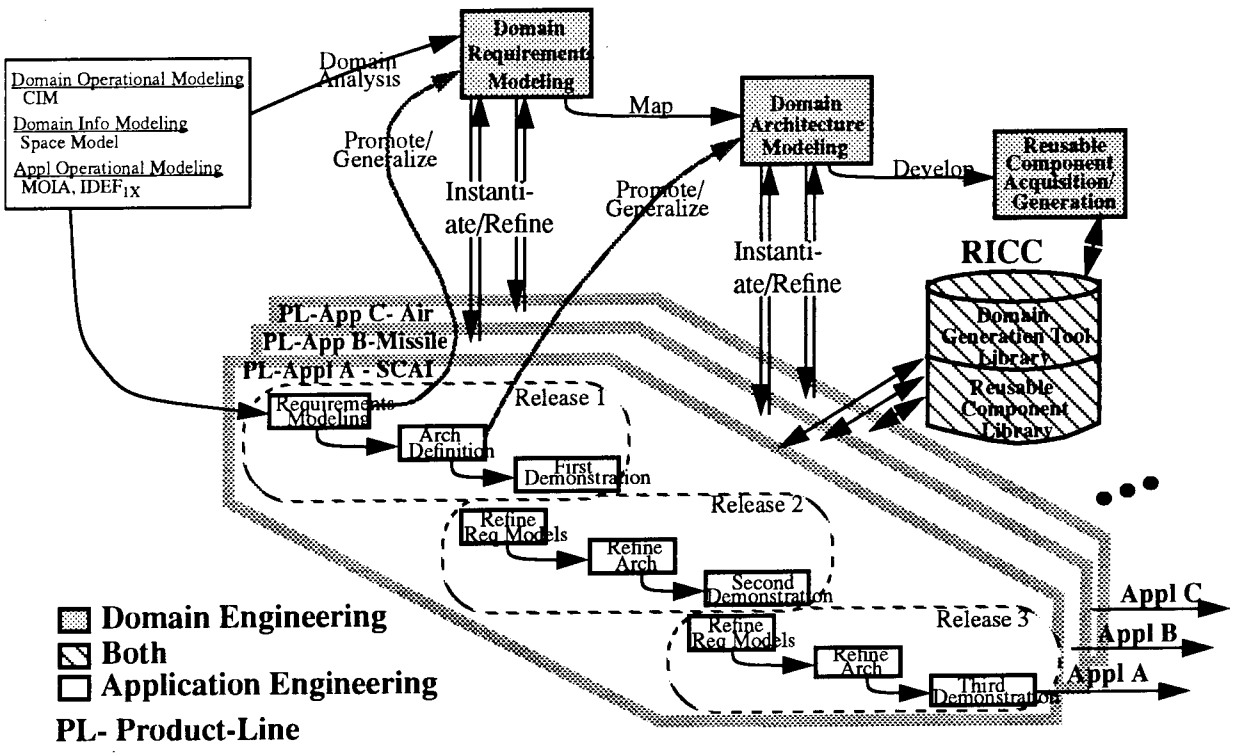


Figure 6. STARS SCAI Two Lifecycle Process

The SCAI DE Approach consists of iterating through the following steps:

- Define the layering scheme for the Engineering Models.
- Define application-level models in support of Application Engineering. (MOIA and $IDEF_{1x}$ models are detailed models to support the development of the DRM.)
- Promote the application-level models to the domain-engineering level after the first SCAI release and generalize them. (The CIM Space Domain Process Model is used to generalize the DRM because its broader scope identifies users as well as systems).
- Refine the domain-level models through a process of iterative instantiation/ refinement during Releases 2 and 3. Continue this process of instantiation/ refinement for follow-on product-line releases.
- Modify future application models output products to support maintenance and continued development of the Two Lifecycle Models.
- As reusable components are validated by use in a particular release, they are saved in a Reusable Component Library. Off-line RICC Program Generators are also viewed as reusable assets.

No claims can be made that this iterative approach to Domain Engineering will work for other domains that are less well understood, or not subject to layering.

(2) High-level progression of engineering artifacts.

As part of capturing the high-level approach, the team developed the general artifact progression summarized in Figure 7. The definitions of the Cleanroom specification volumes can be found in Appendix B. The DE/AE operations models, process models and information models form the foundation for the System level Cleanroom specifications. OO analysis yields the high-level object oriented model which identifies the major classes and objects. Based on project management objectives and based on the coherence of clusters of objects, the OO model is divided into release groupings, corresponding to major incremental releases of the application. For each release, the system specification is refined into a release specification, again using the Cleanroom specification techniques. The release specifications, in turn, launch the release development activities.

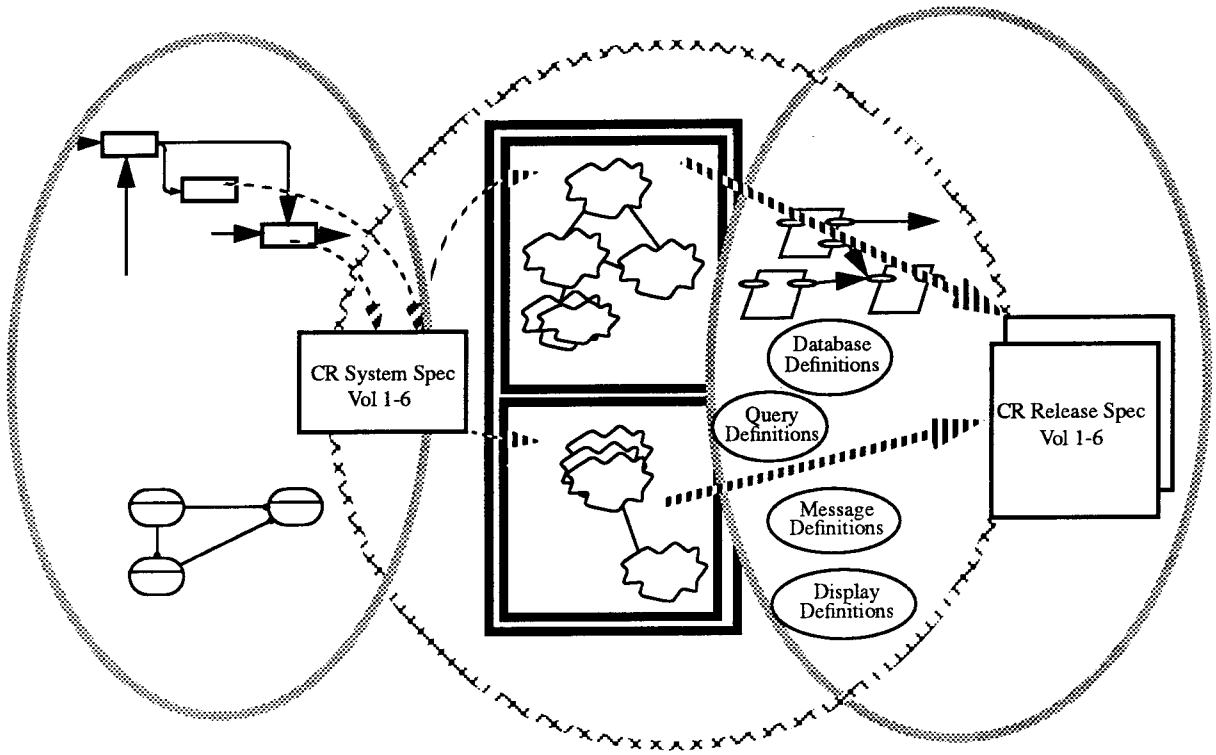


Figure 7. Engineering Artifact Progression

Define/Refine the Process Activity Flow:

Coupling the features of the above three methods/technologies (Cleanroom Software Specification, Booch Object Oriented Analysis, Ada Process Model) resulted in the activities shown in Figure 8, which provides a high level description of the integrated process. Each activity is briefly described below.

(1) Gather System Requirements: Collect information from various sources and scope the system requirements. Resolve requirements issues and clarifications iteratively.

(2) Prepare/Refine Specification: Define the domain/system level requirements. Once the domain/system level requirements are captured, create a Release-level Specification. The focus is on producing a Mission Specification (Vol 1) and a Usage Specification (Vol 2). This activity is coupled with the "Develop/Refine Object Model" activity.

This activity is re-entered (the Refine part) based upon:

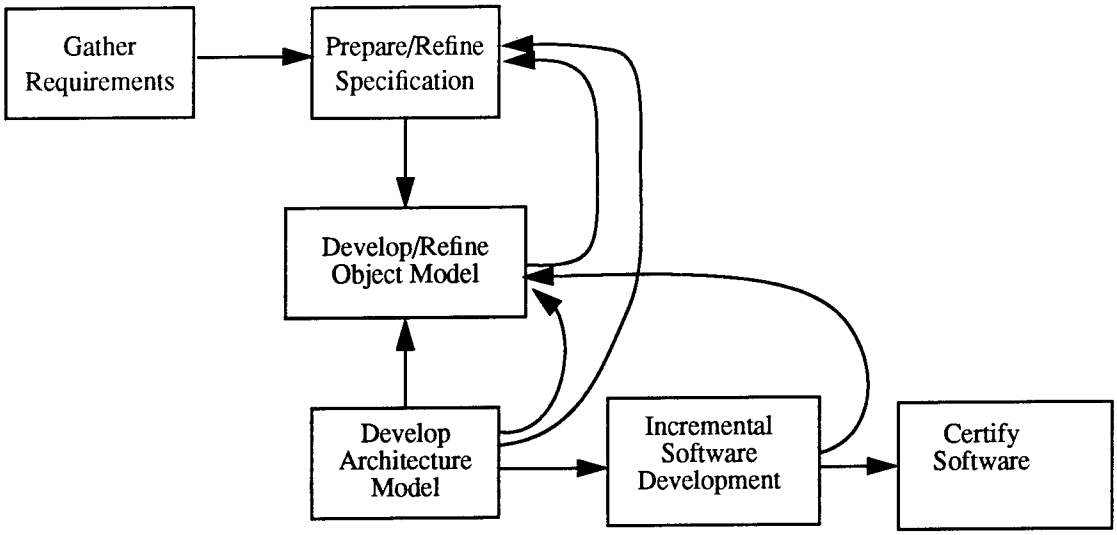


Figure 8. Engineering Workflow

- The Logical View architecture definition. The Cleanroom Black Box Spec (Vol 3) is written based upon this input. The Black Box Specification is validated against the Class Model Object Scenario Diagrams.

- User feedback related to requirements as driven by demonstrations.

Once the Object Model & the Architecture Model are baselined, the Cleanroom Black Box Validation, Usage Profile & Construction Plan (Vol 4-6) are then created.

(3) Develop/Refine Object Model (Logical View of the Architecture): This activity is a Domain Engineering activity. It is presumed that the SWSC mission is to generalize and reengineer existing systems in the SWSC domain. An unabstracted object oriented model of the existing system is created. The model is then abstracted to create a Booch Class Model, or Application Requirements Model. This abstraction activity can be viewed as "reengineering." Or, if a Requirements Model already exists for a different system in the same domain, then the Domain Requirements Model is extended and reabstracted to incorporate the characteristics identified in the unabstracted new system model. The goal of this abstraction activity is to identify the key abstractions (classes) and key mechanisms (scenarios) of the problem domain. This analysis is validated by creating an initial System Architecture Skeleton (SAS - part of the Architecture Model), and executing it, "pumping" data into key scenarios. Next, the class structure is completely elaborated, and any remaining non-essential scenarios are defined. This step is highly iterative with the "Develop Architecture Model" activity. These two activities serve to support the identification of a candidate Domain Architecture Model. At the conclusion of this step, the Domain Requirements Model:

- can be effectively baselined;

- can be used by "Prepare/Refine Specification" as input in creating the Cleanroom Black Box Specification (Vol 3);

- serves as a starting point for the "Incremental Development" activity which will utilize the RICC tools.

(4) Develop/Refine Domain/Application Architecture Model (aka Physical View of the Architecture): Initially, the key abstractions can be used to create a preliminary Architecture Model. As the Object Model solidifies, candidate architectures can be explored to validate key scenarios. A candidate architecture is chosen and preliminary design decisions are demonstrated in a critical thread SAS demo. This demo shows that Quality Performance Requirements (QPRs) can be met in light of critical scenario operations. The chosen Architecture is then baselined. Application SASs must imbed Domain SASs. Developers of Domain SASs should examine the Trade Studies which are recorded as part of the Domain Architecture Model.

(5) Incremental Software Development: Develop displays, messages, database queries and mission components within the structure of the SAS. This activity is iterative with the Develop/Refine Architecture Model activity.

(6) Certify Software: This activity verifies that the developed software exhibits the same behavior as is defined in the specification.

Define/Refine the Engineering Artifacts

The Cleanroom Specification Volumes are defined in three forms: a Domain-level Specification, System-level Specification and Release-level Specification. The Domain-level Specification contains high-level requirements and constraints for the Domain. The System-level Specification contains the high-level system requirements and constraints. Each Release-level Specification contains lower-level requirements as pertaining to a particular release.

The Object Model (aka Logical View of the Architecture) constitutes a logical machine independent design view of the software architecture and consists of:

(1) Class category diagrams, identifying system layers.

(2) Class diagrams depicting the static view of the problem domain which consists of classes & the relationships between classes (class collaboration).

(3) Class specifications for each class that describe the responsibility of each class. Information such as attributes, operations, and relationships are captured textually in the class specification.

(4) Object scenario diagrams that depict the dynamic view of how the classes interact to perform key system mechanisms or threads.

(5) Ada PDL giving an abstract view of Object Behavior.

The Architecture Model (aka Physical View of the Architecture) defines the executable components of the system. A display hierarchy is created using Display Builder. This hierarchy captures the user interface and menu flow. The major functional components of the system are defined using UNAS Tasks. The executable program (called an SAS) consists of UNAS Tasks grouped into UNAS Processes (Equivalent to UNIX Processes) which are allocated to hardware processors. A UNAS Task "withs" in the appropriate Ada packages to carry out its intended functionality.

The mission components are represented as Ada packages. Miscellaneous data files are created and maintained by the various RICC tools. Trade Studies identify different mappings of Classes (Ada Packages) to UNAS tasks, and associated reasons for making decisions leading to the mappings.

Define/Refine the Mapping between the Artifacts

The iterative mapping between the major artifacts is shown in Figure 9. The mapping is shown within the context of the layered framework.

This layering creates a separation of concerns between the framework layers such that:

(1) functional requirements can change, impacting the Object Model but will not necessarily perturb the Architecture Model (including the network topology); and

(2) changes, over time, in the network topology, operating systems, and COTS products won't affect the Object Model or Specification.

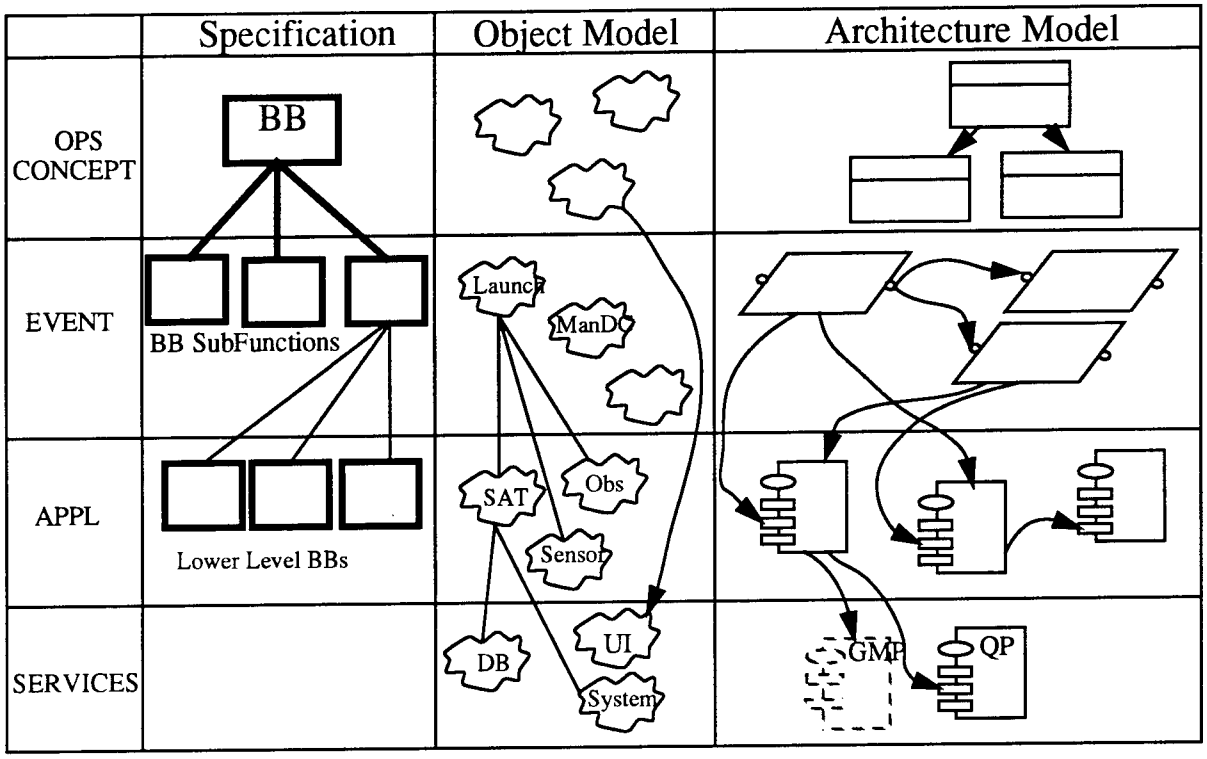


Figure 9. Artifact Mappings

Select/Adjust the Architectural Framework

A key aspect of understanding the artifacts was to define them within the context of an overall architectural framework for C^3I systems. This framework transcends the different artifacts as described below.

The Cleanroom Specification defines a set of black box subfunctions that represent the key system functionality of the system. These black box subfunctions map to the Event Layer which captures the dynamic system behavior.

The overall static object oriented architecture is represented in a layered framework depicted in Figure 9 (middle column). Each layer consists of one or more class categories. The major layers are described as:

(1) Operations Concept Layer: the user behavior derived from MOIA is captured here. The classes consist of sets of user actions (i.e., a session) that perform some functional system behavior. The dynamic interaction of user-computer is described by the use state transition diagrams.

(2) Event Layer: the dynamic system behavior that captures system policies and procedures. This is typically the most volatile part of a system. System events can be things such as satellite launches, or user procedural interactions. The classes at this layer invoke class operations at the Applications Layer to perform system functions.

(3) Applications Layer: where the tangible classes are defined. Things such as system messages are encapsulated. Usually these classes are less dynamic but message classes will encapsulate algorithmic processing. The stable, static system data is encapsulated at this level. In essence, these classes form an object layer over lower-level services such as UNAS, Query Processor & Display Builder packages.

(4) Services Layer: solution space services such as the RICC components.

The Architecture Model (aka Physical View of the application) consists of the same architectural layers as described above which are built upon the RICC components. These are:

(1) Operations Concept Layer captures the user interactions of the system. All screens and user actions are represented here. The RICC Display Builder tool creates artifacts (Ada main programs, etc.) that reside in this layer.

(2) Event Layer in which dynamic system behavior is captured in a Software Architecture Skeleton (SAS) developed using the UNAS tool. UNAS Tasks are used to capture dynamic system behavior. A system architect defines the tasks based upon the Object Model. The system architect also defines the physical layout (network topology) of the architecture. UNAS Tasks are allocated to UNAS Processes and hardware processors in this step.

(3) Application Layer which is an object layer (implemented as Ada packages) that represent information storage & retrieval while encapsulating system data. A key approach in this layer is to build an object layer on top of the Service Layer (RICC components) such that a class (its data & operations) defined by the Object Model is mapped into Ada package specs. The implementation of the class (via class specific code & calls to RICC services) is hidden in the package body.

(4) Services Layer which contains solution space services such as the RICC components.

Detail /Improve the Process

This captures the process to the level needed for practitioners who are using it to build product-line applications. Rather than attempt to define the entire process at once, an incremental approach is followed: modest portions are chosen for detailing, and the experience gained in executing the selected portions is used to guide further definition activities.

Use the Defined Process

The process definition increment concludes with its most important part: use for real project activities and feedback to help improve the process and to guide process definition activities in related areas.

Appendix F

SEE User Surveying Experience

In an attempt to better understand the users' perceptions of the SEE, with the ultimate goal of improving the SEE, the Air Force/STARS Demonstration Project believes in conducting SEE user surveys. In the past year, two SEE user surveys were conducted. This appendix presents a summary of each survey and presents a more detailed discussion of two SEE survey innovations and the tentative conclusions derived from their results. The actual surveys themselves and the results will not be presented in this appendix. Readers wishing to obtain copies of the surveys or additional information may contact Capt. Scott Kent, SWSC/SMXM, at (719)554-9082 or Richard Randall, Loral/FS at (719)554-6597.

Survey 1 - Summary Results

The first user survey was conducted in December of 1993 by asking a series of 21 questions during one-on-one interviews with 13 SEE users (out of 20 possible users). The survey was subdivided into 5 categories: demographics, usability, change management, training/coaching, and overall environment. Representatives from each contractor (CACI, Kaman Sciences, Rational, TRW) and the demonstration partners (AF and Loral) were selected to participate in the survey. Overall, the results of the survey were favorable, more so than expected, with the following specific observations:

(1) There were no significant differences in the responses between the various contractors (CACI, Kaman Sciences, Rational, TRW) and the partners (AF and Loral).

(2) Users had enough time to learn the capabilities of the SEE.

(3) More AIX fundamentals training was needed, specifically concerning use of man pages, system accounting, AIXWindow defaults, and login customizations.

(4) The process for suggesting and implementing changes to the SEE could be improved to increase user confidence.

Survey 2 - Summary Results

The second survey was conducted in August 1994 and was administered in a different manner than the first; instead of conducting personal interviews, a paper copy of the survey was given to 12 SEE users (out of 35 possible users) to fill out at their leisure. As with the first survey, representatives from each contractor and the partners were selected to participate in the survey. The format of the second survey is also different than the first survey. There are a series of 16 questions subdivided into 4 categories: demographics, general SEE questions, training/coaching, and problem reporting/problem resolution.

In addition to the 16 questions, users were asked to fill out a tools matrix, and a SEE Integration diagram. The tools matrix was introduced to obtain specific feedback concerning the individual tools currently available on the SEE. The SEE Integration diagram was introduced to learn what the users believed to be high payoff SEE integration opportunities. Both of these sections are explained in the survey innovation section below.

Overall, the results of the second survey were very positive, once again more so than expected. Below are some specific observations:

(1) The SEE had the required functionality.

(2) Users felt they got adequate resolution of SEE problems both in timeliness and in level of resolution.

(3) SEE performance was marginal and SEE tool and system response times needed improvement.

SEE Survey Innovations

The Demonstration Project, in conducting SEE surveys, feels it has come up with two survey innovations that may be of interest to projects considering similar surveys. One uses an evaluation matrix to assess the suitability of specific SEE tools; the other uses a diagram of tool functionality clusters to assess the high-payoff opportunities for improving SEE integration. The following discusses each technique and the tentative conclusions that have been formed by analyzing the survey results. Also discussed are deficiencies in the survey that have been disclosed by our analysis. These deficiencies will be taken into account for the next survey, scheduled for early next year.

(1) **Tool Evaluation Matrix**

The matrix shown in Figure 1 was used to collect data on users' perceptions of the tools they used (the figure shows a sample of one user's matrix results.) Users were asked to assess the importance of individual tools to their work and then to assess the tool's current usability. They were further asked to assess the potential of the tool to the long-range project objectives. The responses thus help SEE engineers evaluate both the present status of the SEE and the role of individual tools in SEE evolution.

Figure 2 shows the consolidated matrix results, formed by combining the data from all 12 user surveys.

Two analysis techniques were used to form preliminary conclusions about the tools:

- **Simple consideration of rankings**

  Judgements were based on the averages of the Current and Future Suitability rankings, regardless of the number of respondents that ranked the tools or how important the tools were. Using this method, the results were as follows:

  - Current suitability:

    Questionable: Uniplex, SODA, Project Catalyst, CAT Compass are between Not suitable and Moderately Suitable

    Strong: APEX, Frame, RICC QB, RICC MSG, ROSE, SALE are between Moderate and Very

  - Long-term potential

    Questionable: CAT Compass, Exclaim, Project Catalyst, ProDAT, and Uniplex are between Low and Some Value

    Strong: Amadeus, APEX, Frame, RICC DB, QB, MSG, ROSE, SODA are between Some and High Value.

Tool Questionnaire

For each of the tools listed, please circle one number in each of the three categories: Importance, Current Suitability, and Long-term Potential.
(1) Importance: In this section, indicate how important this tool is to you in getting your job done.
(2) Suitability: In this section, indicate how suitable the tool is for your job. Take all factors into account (usability, functionality, reliability).
(3) Potential: In this section, indicate what value the tool has in terms of SEE strategy (to help guide future investment).
Comments and recommendations are encouraged; place them in the Comments/Recommendations column.

Tool	(1) Importance (circle one each) 0 = Don't use 1 = Low 2 = Medium 3 = High				(2) Current Suitability (circle one each) 1 = Not Suitable 2 = Moderately Suitable 3 = Very Suitable			(3) Long-term Potential (circle one each) 1 = Low value 2 = Some value 3 = High value			Comments/Recommendations
Amadeus	0	1	2	3	1	2	3	1	2	3	
Apex	0	1	2	3	1	2	3	1	2	3	- Get integrated Verdix/APEX ASAP.
Cat Compass	0	1	2	3	1	2	3	1	2	3	
CMVC/6000	0	1	2	3	1	2	3	1	2	3	
Design IDEF	0	1	2	3	1	2	3	1	2	3	
Exclaim	0	1	2	3	1	2	3	1	2	3	
Fortran	0	1	2	3	1	2	3	1	2	3	- SPADOC specific, may not be generally needed
FrameMaker	0	1	2	3	1	2	3	1	2	3	
Integrator	0	1	2	3	1	2	3	1	2	3	
Process Weaver	0	1	2	3	1	2	3	1	2	3	
Project Catalyst	0	1	2	3	1	2	3	1	2	3	
PRODAT	0	1	2	3	1	2	3	1	2	3	
RICC (Display Builder)	0	1	2	3	1	2	3	1	2	3	
RICC (Query Builder)	0	1	2	3	1	2	3	1	2	3	
RICC (X-msg Tool)	0	1	2	3	1	2	3	1	2	3	
ROSE	0	1	2	3	1	2	3	1	2	3	
SODA	0	1	2	3	1	2	3	1	2	3	
SPMS	0	1	2	3	1	2	3	1	2	3	
SALE	0	1	2	3	1	2	3	1	2	3	
Teamwork	0	1	2	3	1	2	3	1	2	3	
Uniplex	0	1	2	3	1	2	3	1	2	3	
Verdix	0	1	2	3	1	2	3	1	2	3	- Merging with APEX
Workbench	0	1	2	3	1	2	3	1	2	3	

Figure 1. One User's Response to Tool Questionnaire

For each of the tools listed, please circle one number in each of the three categories: Importance, Current Suitability, and Long-term Potential.
(1) Importance: In this section, indicate how important this tool is to you in getting your job done.
(2) Suitability: In this section, indicate how suitable the tool is for your job. Take all factors into account (usability, functionality, reliability).
(3) Potential: In this section, indicate what value the tool has in terms of SEE strategy (to help guide future investment).
Comments and recommendations are encouraged; place them in the Comments/Recommendations column.

Tool	(1) Importance (circle one each) 0 = Don't use 1 = Low 2 = Medium 3 = High				(2) Current Suitability (circle one each) 1 = Not Suitable 2 = Moderately Suitable 3 = Very Suitable			(3) Long-term Potential (circle one each) 1 = Low value 2 = Some value 3 = High value			Comments/Recommendations
	0	1	2	3	1	2	3	1	2	3	
Amadeus	5	2	2	3	0	6	1	0	2	5	
Apex	7	1	0	4	0	2	4	0	2	7	- Get integrated Verdix/APEX ASAP. - Slow!
Cat Compass	8	3	1	0	2	2	0	4	1	1	
CMVC/6000	0	4	2	6	0	10	2	1	6	4	- Need more guidelines on usage
Design IDEF	7	1	2	2	0	3	2	0	5	2	
Exclaim	10	0	2	0	0	1	1	3	1	0	- Is there a better spreadsheet?
Fortran	10	0	1	1	0	1	1	2	0	2	- SPADOC-specific, may not be generally needed
FrameMaker	0	1	2	9	0	4	8	0	6	6	
Integrator	11	0	1	0	0	1	0	0	2	1	
Process Weaver	8	2	1	1	1	3	0	0	4	1	- Very time consuming
Project Catalyst	7	2	1	2	3	2	0	1	6	0	- Suitable with workarounds
PRODAT	11	1	0	0	0	1	0	1	4	0	
RICC (Display Builder)	7	1	1	3	0	2	2	0	4	4	- Needs more capability
RICC (Query Builder)	8	0	0	4	0	0	3	0	2	5	- Could have better user interface
RICC (X-msg Tool)	9	0	0	3	0	0	2	0	2	4	
ROSE	8	0	3	1	0	1	2	0	4	4	- Will be better when Rose/Ada is out
SODA	9	1	1	1	2	1	0	0	3	3	- Has much potential
SPMS	7	2	1	2	0	5	0	0	5	2	
SALE	9	0	0	3	0	0	2	0	2	4	
Teamwork	9	0	1	2	1	0	2	1	1	3	
Uniplex	5	5	2	0	5	2	0	6	1	0	- Need a better product
Verdix	8	0	1	3	0	2	2	1	4	3	- Merging with APEX
Workbench	7	1	3	1	0	4	1	1	3	1	- Eventually move to Tooltalk

Figure 2. Consolidated Tool Questionnaire Results

- Weighted rankings

  In this case, a weighted metric was determined for each tool by multiplying the importance rankings by the average suitability ranking - and similarly by the average potential ranking. This yielded the metrics shown in Table 1:

SEE Tool[a]	# Users Responding	Average Importance (I)	Average Suitability (S)	Average Potential (P)	Weighted Suitability (I * S)	Weighted Potential (I * P)
Amadeus	7	2.0	2.1	2.7	4.2	5.4
APEX	6	2.6	2.6	2.8	6.8	7.2
CAT/Compass	4	1.2	1.5	1.5	1.8	1.8
CMVC/6000	12	2.1	2.1	2.2	4.4	4.6
Design IDEF	5	2.2	2.4	2.2	5.3	4.8
Exclaim	2	2.0	2.5	1.2	5.0	2.4
Fortran	2	2.5	2.5	2.0	6.3	5.0
Frame	12	2.6	2.7	2.5	6.9	6.5
Integrator	1	2.0	2.0	2.3	4.0	4.6
Process Weaver	4	1.7	1.7	2.2	2.9	3.7
Project Catalyst	5	2.0	1.4	1.8	2.8	3.6
RICC DB	4	2.2	2.5	2.5	5.5	5.5
RICC MSG	2	3.0	3.0	2.6	9.0	7.8
RICC QB	3	3.0	3.0	2.7	9.0	8.1
ROSE	3	2.2	2.6	2.5	5.7	5.5
SALE	2	3.0	3.0	2.3	9.0	6.9
SODA	3	2.0	1.3	2.5	2.6	5.0
SPMS	5	2.0	2.0	2.2	4.0	4.4
Teamwork	3	2.6	2.3	2.4	6.0	6.2
Uniplex	7	1.2	1.2	1.1	1.4	1.3
Verdix	4	2.5	2.5	2.0	6.3	5.0
Workbench	5	2.0	2.2	2.0	4.4	4.0

TABLE 1. Weighted Rankings of Individual SEE Tools

a. (Note: data on ProDAT is omitted, because experience with the tool was insufficient at the time of the survey)

Though the data is incomplete (see below for specifics), some tentative conclusions are possible.

- The following tools have a "weighted potential" metric above the average and thus can be viewed as strategically important to their users: Amadeus (metrics), APEX and Verdix (Ada development), Design IDEF (process definition), FORTRAN (reverse engineering of existing systems), Frame and SODA (documentation production), RICC tools and SALE (architectural infrastructure and code generation), ROSE and Teamwork (engineering modeling), SODA (document production). This seems to suggest that the organization should pay special attention to the evolution of these tools - and that SEE integration attention would not be wasted on these tools, since they are less likely to be replaced with other tools.

- SODA (a documentation production tool) needs work: it is viewed as strategically important, but it falls short in its current implementation. Note that this is a very new tool; and Rational is actively pursuing it as an important part of its product strategy and has already released an improved version since the time of the survey.

In retrospect, there are some deficiencies with the data collected, and these observations will be taken into account in the next survey:

- The actual number of users of each tool is missing. This metric would be useful in assessing overall importance, since the more people whose jobs depend on the tool the more benefit realized by improving the tool.
- The reason for a tool's unsuitability (or strength) is missing. This was intentionally left out of the survey due to time pressure, but it would help in clarifying responses and in giving vendors more specific feedback about their products.
- The importance of each "functionality cluster" to users is missing. Thus, although an individual tool might be viewed as *low* in potential, it is possible that the functionality represented by the tool is *high* in importance - perhaps calling for an entirely new tool. (The next section, on SEE integration, offers some additional insight on this topic, since feedback on functionality clusters was requested in that section.)

It should also be noted that since the survey was taken, there have been updates to several tools that have improved their functionality and behavior. SODA, SPMS, ProjectCatalyst, and CAT/Compass are all examples: all of these were Beta test tools during the evaluation period.

(2) **SEE Integration Evaluation Diagram**

This diagram was intended to obtain insight into what the users feel are high payoff opportunities for SEE integration. The diagram shows "functionality clusters" arranged in a circle. The users were instructed to consider the current level of integration (left loosely defined) among the clusters and to indicate where the biggest payoffs would be for improving that integration. If further integration would significantly improve the SEE's ability to support the user's job responsibilities, the user was to recommend this by drawing lines between pairs of clusters. The functionality clusters identified for the survey are shown below (with current SEE tools shown in parentheses to help define what the clusters were):

- Object Oriented and Information Modeling (ROSE, Teamwork/IM)
- Infrastructure Tools (RICC Tools, SALE)
- Ada Software Implementation (Apex, Verdix, RCI, Ada Analyzer)
- Software Certification
- Reuse Management
- Metrics (Amadeus, SPMS)
- Process Enactment (Project Catalyst, Process Weaver)
- Process Modeling (Design/IDEF, SPMS, FrameMaker)
- Project Management (CAT/Compass)
- Configuration Management (CMVC/6000, Apex/CMVC)
- General Office Support (Mail, Uniplex, Frame, Exclaim)
- Documentation Production (Frame, SoDA)

Figure 3 illustrates the symbology used for this diagram. A line between clusters specifies a need for improved integration; a thick line specifies an especially high payoff opportunity for improved integration. A line from one cluster to itself indicates that integration should be improved among the tools within the cluster.

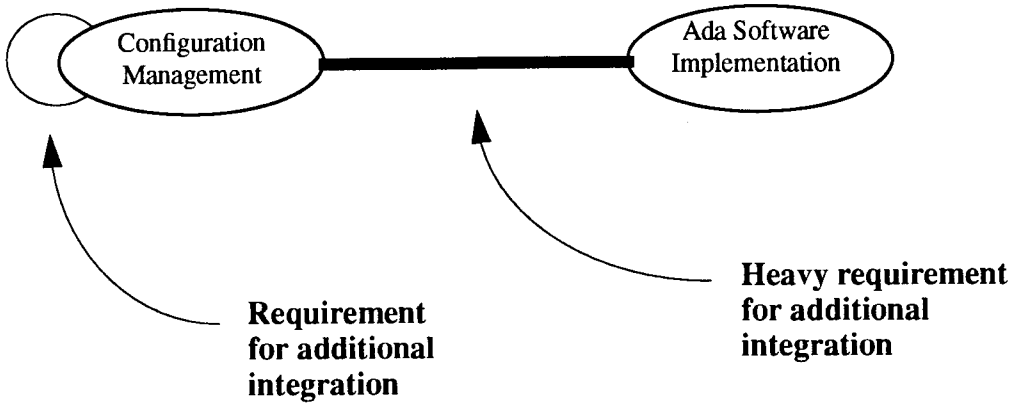


Figure 3. Symbology used on Tool Integration Diagram

Figure 4 shows a sample completed diagram with one user's responses. The diagram shows the "functionality clusters" as ovals arranged in a circular pattern around the page. To clarify what was meant by each cluster, the relevant tools in the current SEE were listed next to each cluster's oval.

The completed diagrams showed that users were willing to give this analysis careful consideration. Their responses were combined onto one diagram, shown in Figure 5, with arcs between clusters assigned a metric based on the number of users recommending further integration. The metric was computed by adding up the lines between clusters, counting one for each *thin* line and two for each *thick* line.

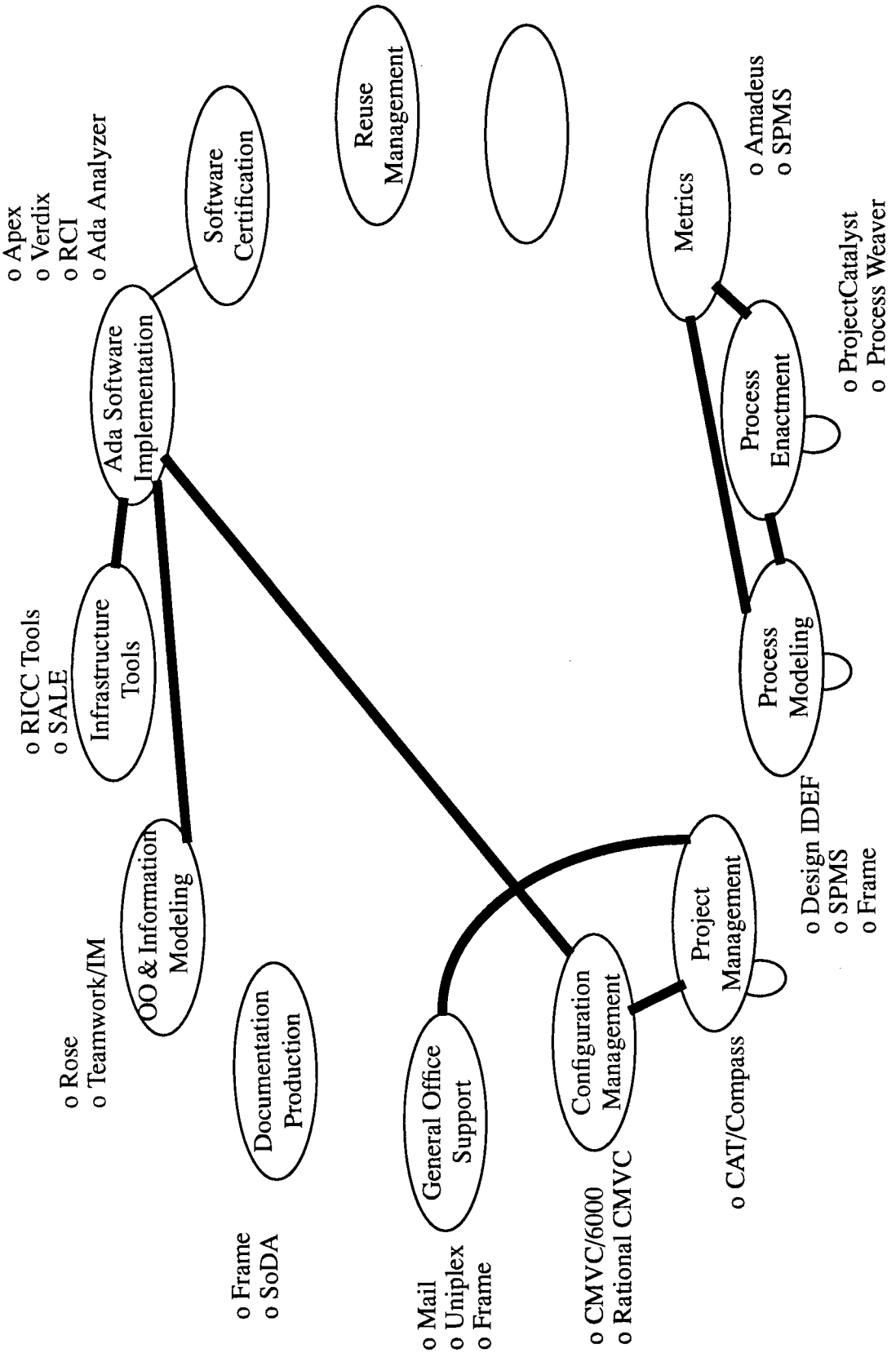


Figure 4. -- High Payoff Integration Opportunities(
(From the Perspective of a Sample User's Job Responsibilities)

SEE Integration Areas of Payoff

-- This diagram represents the combined user responses to the SEE Integration question. Each line is tagged with a value. The value was obtained by taking each user's diagram and assigning one point to each thin line and two points to each thick line. All of the points for each line between functionality clusters were summed and are depicted as the values on the lines below.

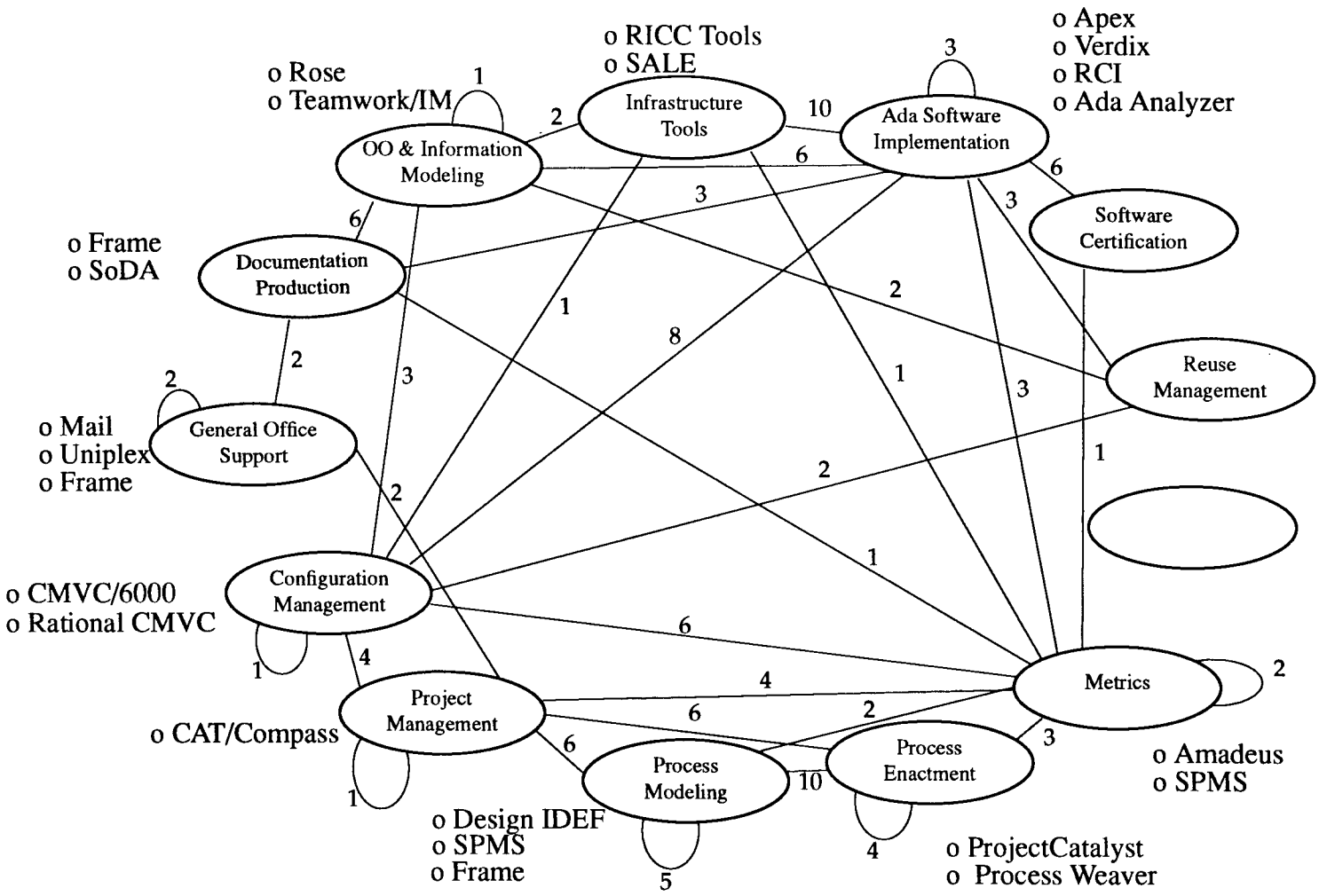


Figure 5. High Payoff SEE Integration Opportunities (Consolidated Responses of All Users)

These results were considered from two different standpoints:

- **Inter-Cluster Patterns**

  A scan of the consolidated results diagram (Figure 5) suggested that improved integration among the following cluster groupings will be strategically important - with exceptionally strong requirements flagged with "*":

  - * Configuration Management and most other clusters
  - * Project Management and most other clusters
  - Metrics and most other clusters
  - * OO & Information Modeling, Document Production, and Ada Software Implementation
  - Ada Software Implementation, Infrastructure Tools, and Software Certification
  - Process Modeling, Project Management, and Process Enactment

- **Single Cluster Focal Points**

  A metric was computed for each cluster by adding the numbers on connections emanating from the cluster's oval. Clusters with high metrics may be good candidates for integration focus. Table 2 shows these metrics, and suggests that Ada Software Development, Configuration Management, Metrics, Process Enactment, Process Modeling, and Project Management are high-payoff integration focal points.

Functionality Cluster	Integration Need Metric (see text)
OO & Information Modeling	20
Infrastructure Tools	14
Ada Software Development	42
Software Certification	7
Reuse	7
Metrics	23
Process Enactment	23
Process Modeling	23
Project Management	21
Configuration Management	25
General Office Support	6
Document Production	11

TABLE 2. "Single Cluster Focus" Integration Potential

As in the other part of the survey (the individual tool evaluations), analysis disclosed survey deficiencies that must be taken into account for the next survey, including:

- Users' perceptions of the current level of integration among the clusters is missing (this was intentionally left out in an effort to reduce the impact of the survey to respondents, since the survey timeframe was short).
- Users were not asked to rate the importance of each functionality cluster to their jobs.
- Users were not asked to rate the strategic importance of each cluster to the long-range product-line objectives.
- The number of people using each functionality cluster is missing.

Planning for the Next Survey

The next survey is planned for early next year. In preparing for the survey, the following will be considered:

(1) **Number of survey participants**: Target at least 20 participants (vs. the 12 participants for this survey). It would be useful if everyone could take the survey, but we will probably once again need to select a representative cross-section. If possible, however, certain information should be requested of each project member:

- Which tools are important to your job? How important?
- Which tools do you consider strategically important to long-range product-line objectives? How important?
- Which functionality clusters are important to your job? How important?
- Which functionality clusters do you consider strategically important to long-range product-line objectives? How important?

Thus, two survey renditions may be appropriate: a detailed survey for a representative subset, and an abbreviated survey for the rest of the user population.

(2) **Gather additional information**: Adjust the survey to gather other key information (as discussed in earlier sections).

(3) **Determine an additional metric to judge long-range "product-line" importance**: Users are being asked to respond to the survey from the point of view of their own job discipline(s). It may be useful to have a small number of managers and key product members assess the overall long-range importance of each functionality cluster.

(4) **Obtain feedback on how to improve the survey**: A copy of this report will be distributed to the project at large, and feedback received will be used to improve the next survey. In addition, the next survey will include a section allowing users to give their feedback on the content and conduct of the survey, as well as feedback on how well this report reflected their input from the prior survey.

(5) **Improve CMVC/6000 process for collecting tool experience**: A SEE improvement process improvement is planned to allow users to document problems with tools - not for the purpose of requesting local system management action, but simply for the purpose of collecting information about tools for feedback to the vendors.

(6) **Obtain outside survey consulting**: Consultation with several outside sources will also be sought, including:

- Dr. John Bailey (from IDA, a STARS program metrics consultant), and
- Maj. Heintzelman, who is the SWSC/SMX officer in charge of metrics.

Appendix G

Megaprogramming and SEE Integration

The material in this section is intended to supplement Section 4.0, SEE Support, in the main body of the Experience Report. It consists of an excerpt from a paper entitled "Integrating a SEE for Megaprogramming" [RandallA95], to be presented at the Software Technology Conference, STC '95.

The excerpted section discusses the rationale for viewing a megaprogramming organization's SEE as a product-line of its own - which parallels the organization's application product-line. It goes on to discuss how megaprogramming principles can be applied to engineering and maintaining a SEE.

1. MEGAPROGRAMMING AND SEE INTEGRATION

1.1 VIEWING THE SEE AS A MEGAPROGRAMMING PRODUCT LINE

Although this paper is primarily an experience report, the authors' reflection has led us to an interesting extension of the megaprogramming approach - an extension that has helped us to better understand our own Demonstration Project experience, and one that may be of use to others in plotting the course for their SEE. The realization is this: ***the SEE itself is a product-line and is as much subject to megaprogramming principles as the applications it supports.***

Consider that the fundamental intent of megaprogramming is to enable systematic reuse across a product-line of applications - applications that are under the control of a single domain manager, and whose natures are similar enough to merit a common engineering approach. As shown in Figure 1, it is natural to think of the organization's SEE assets as several ***logical*** SEEs - even if they share many of the same ***physical*** assets (workstations, software licenses, network connections, etc.). In effect, each application has its own supporting SEE.

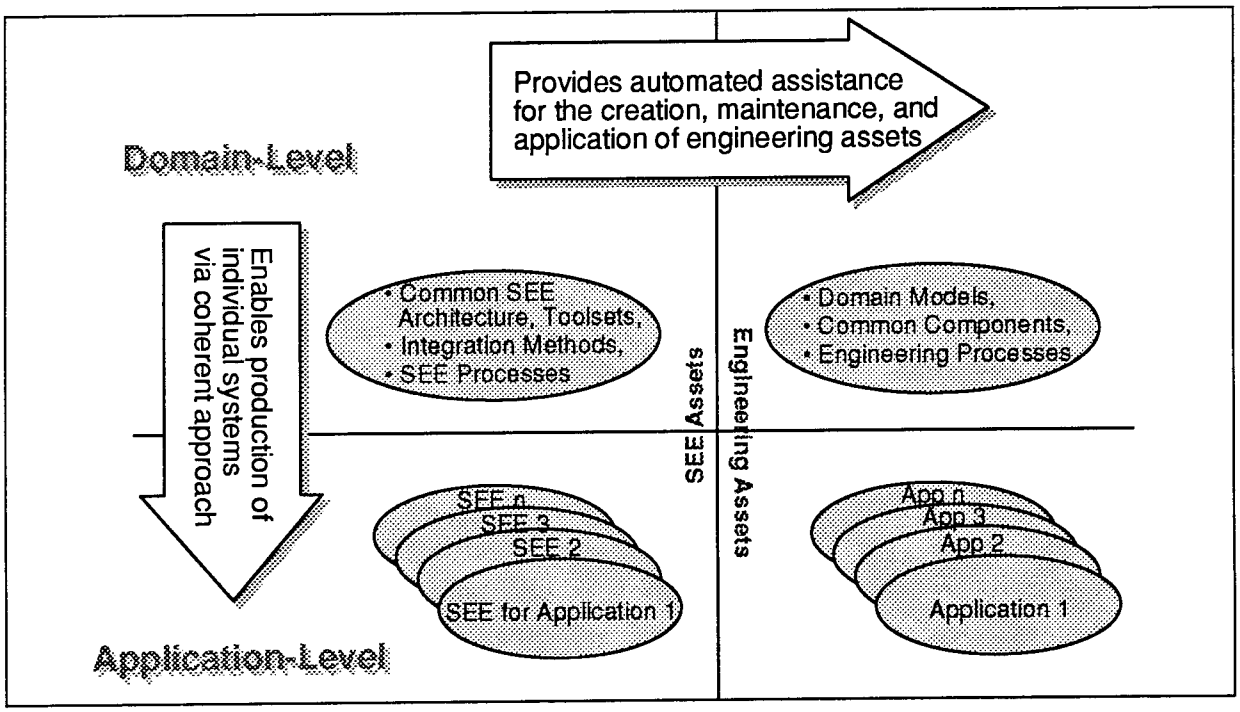


Figure 1. A Product-Line of SEEs in Support of a Product-Line of Applications

Although we want the maximum commonality across this group of SEEs - just as we do with a product-line of applications - we also acknowledge that there are unique aspects of each. For example, for a SEE supporting a space catalog maintenance application, we may need capabilities which are not needed for a missile warning

application - such as a special orbital mechanics analysis support package, or a Prolog engine to support rule-based artificial intelligence research. In addition, although the megaprogramming processes for each application follow a common process architecture, each application has its own nuances - thus each SEE must support a tailored rendition of the common process.

1.2 APPLYING MEGAPROGRAMMING DISCIPLINE TO THE SEE

Since there does appear to be a product-line of SEEs, the natural question is: do megaprogramming principles apply to developing, maintaining, and evolving the product-line? Although space does not permit a detailed analysis here, the answer appears to be yes, as illustrated in Table 1.

Megaprogramming Principle	Applicability to SEE Product-Line
• Domain-Specific Reuse	• Domain requirements established by ***Application Product-line's*** common process framework and need for automated support; domain engineering is appropriate to establish domain models, etc., for the ***SEE Product-Line.***
	• Common SEE architecture framework (see Table 2) to maximize potential for reuse
	• Common artifact management strategy, CM strategy (including reuse library)
	• Common toolset, minimizing license costs, minimizing training costs, enabling maximum use of common integration
	• Common toolset tailoring and integration approach, with process for product-line adaptation of common scripts and data
• Systematic Process	• View application product-line's processes as integral to the SEE
	• Use formal processes for SEE procurement, SEE engineering, SEE assembly and installation, SEE management (problem and work item tracking) - and SEE improvement (including survey techniques)
• Automated Support	• Automated tracking tool to assist with SEE administration
	• Commercial GUI builder for locally-written applications
	• Commercial integration framework services

Table 1. Applicability of Megaprogramming Principles to SEE Product-Line

To reiterate, although this discussion shows that a product-line mentality is highly appropriate to the SEE product-line, it is only on reflection that our group has come to this realization. For example, we did not start the Demonstration Project with a concerted effort to plan out our SEE using domain engineering. As it turns out, however, we now can see that we have been using product-line thinking as we decided what hardware to acquire and which tools were strategic.

For example, Table 2 illustrates how our current SEE exhibits architectural characteristics that provide a good foundation for domain-specific reuse for the SEE product-line.

Architectural Characteristics	Example Sub-Categories	SCAI SEE Domain Implementation Examples
• Layered architecture	• Common underlying operating system environment	• Unix, TCP/IP, NFS, X
	• Process support environment	• STARS PSE
	• Major functionality encapsulations	• Rational's integration family of Ada support tools (centered around Apex)
		• TRW's integration family of code generation tools supporting the architectural infrastructure (UNAS, RICC)
• Common user interface	• Common window-handling	• Motif
	• Common window behavior look and feel	• Open Interface (from Neuron Data), Display Builder (from TRW)
• Common program interface mechanisms	• Low-level API services	• Broadcast Message System (part of HP's SoftBench)
		• TCP/IP Sockets (for guaranteed message delivery)
	• High-level data repository services	• COTS DBMSs (Oracle, Sybase)
• Component reuse	• COTS tools	• Various

Table 2. Architecture Characteristics for the SEE Product-Line

Our intent here is not to say we have done a remarkable job of megaprogramming for the SEE (after all, we are only supporting a single application so far), but rather to say we believe megaprogramming applies to this product line.

As we learn more about how to do megaprogramming, we look forward to applying the improved techniques and practices to the SEE product-line as well.

1.3 A KEY INTEGRATION LAYER: THE PROCESS SUPPORT ENVIRONMENT

Figure 2 illustrates how each of the three megaprogramming technology thrusts is brought to bear to engineer the product-line SEE. The SEE is depicted in the center of the diagram as two layers - a key architectural viewpoint that is elaborated below.

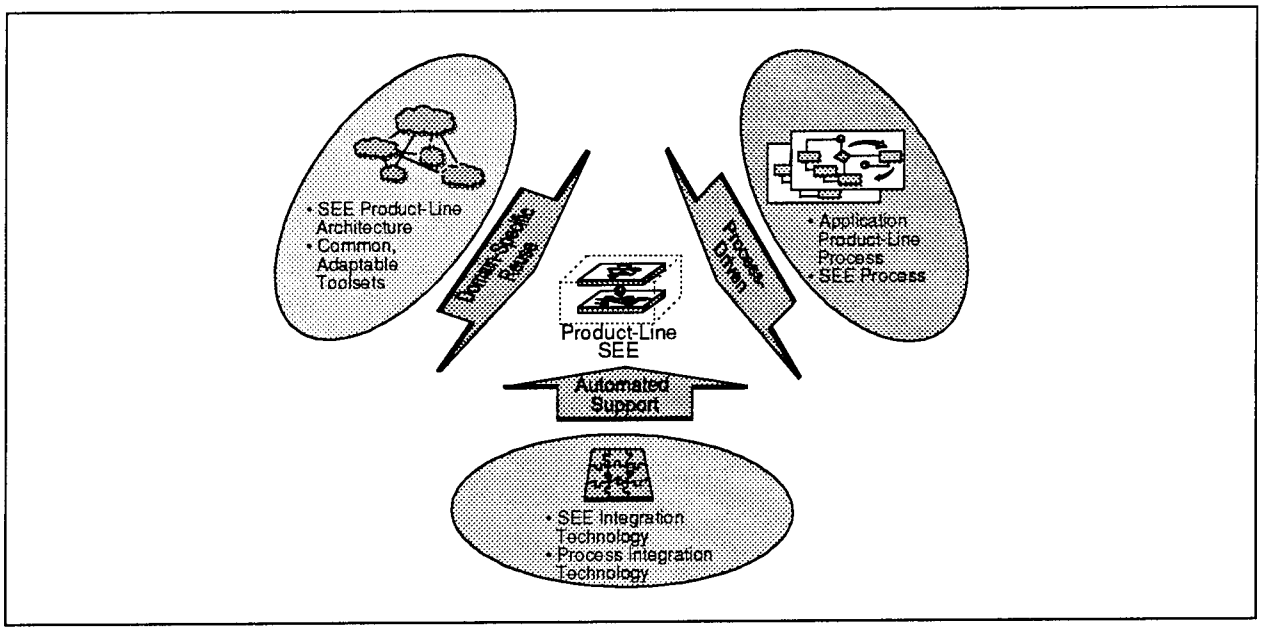


Figure 2. The Motivation for a PSE Encapsulation Layer

A key consequence of this line of reasoning is the identification of an "encapsulation layer" that can be instrumental in SEE integration: the Process Support Environment (PSE). As depicted in Figure 3, the PSE is a set of tools and associated data that allows an organization to define, carry out, and improve its process. It also serves as an integration vehicle for orchestrating the rest of the SEE.

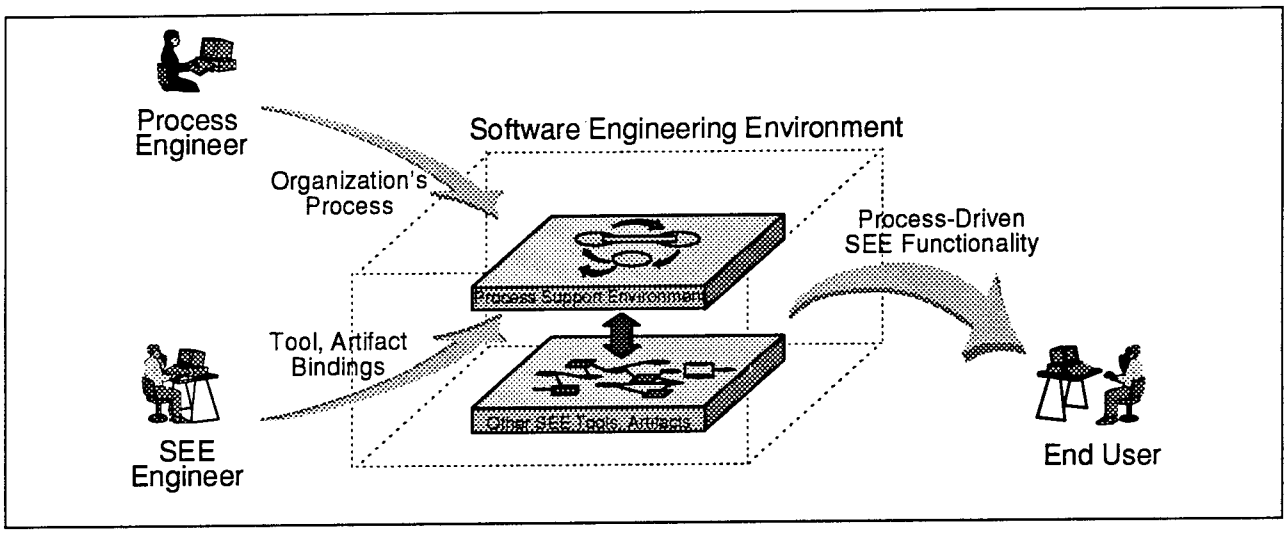


Figure 3. Conceptual Model for a Process Support Environment Encapsulation Layer

The result is a unique form of integration: process is integrated with the SEE, ***and*** the encoded process is then integrated with the rest of the SEE's tools and artifacts. Thus, ***two types of integration are occurring simultaneously***.

Figure 4 shows how the PSE layer fits in context with an abstract architecture for the SEE - depicted in terms of several other possible encapsulation layers that can be called upon to carry out process functions.

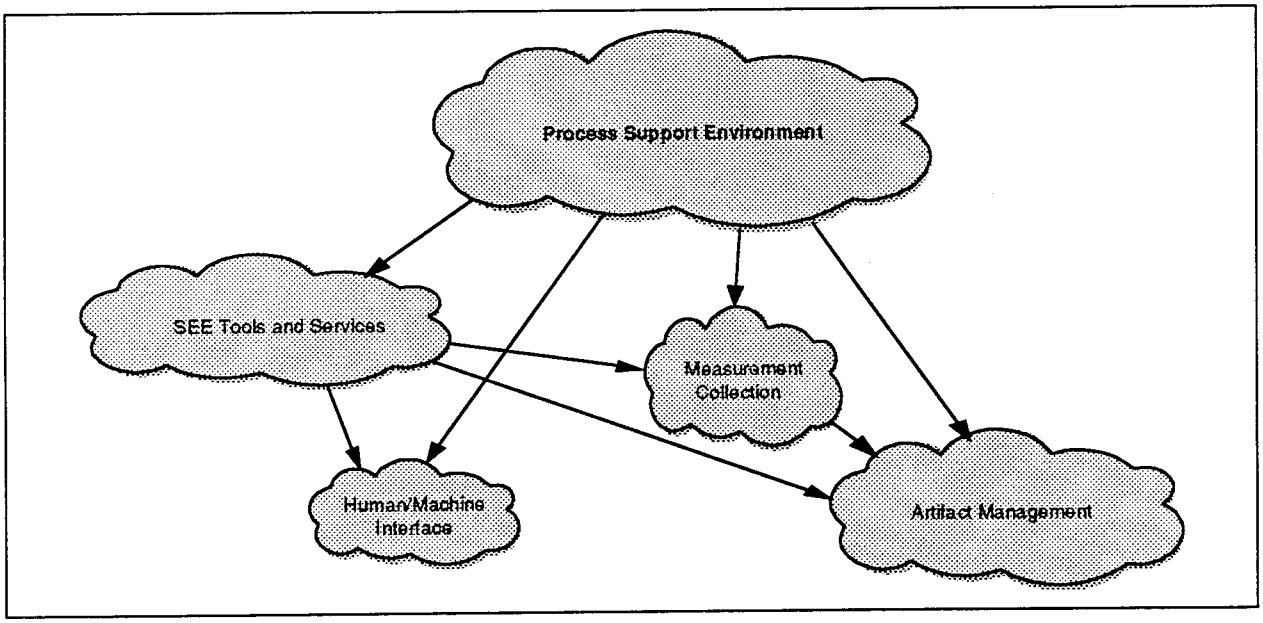


Figure 4. SEE Encapsulation Layers

Figure 5 illustrates the PSE currently in use on the Demonstration Project.

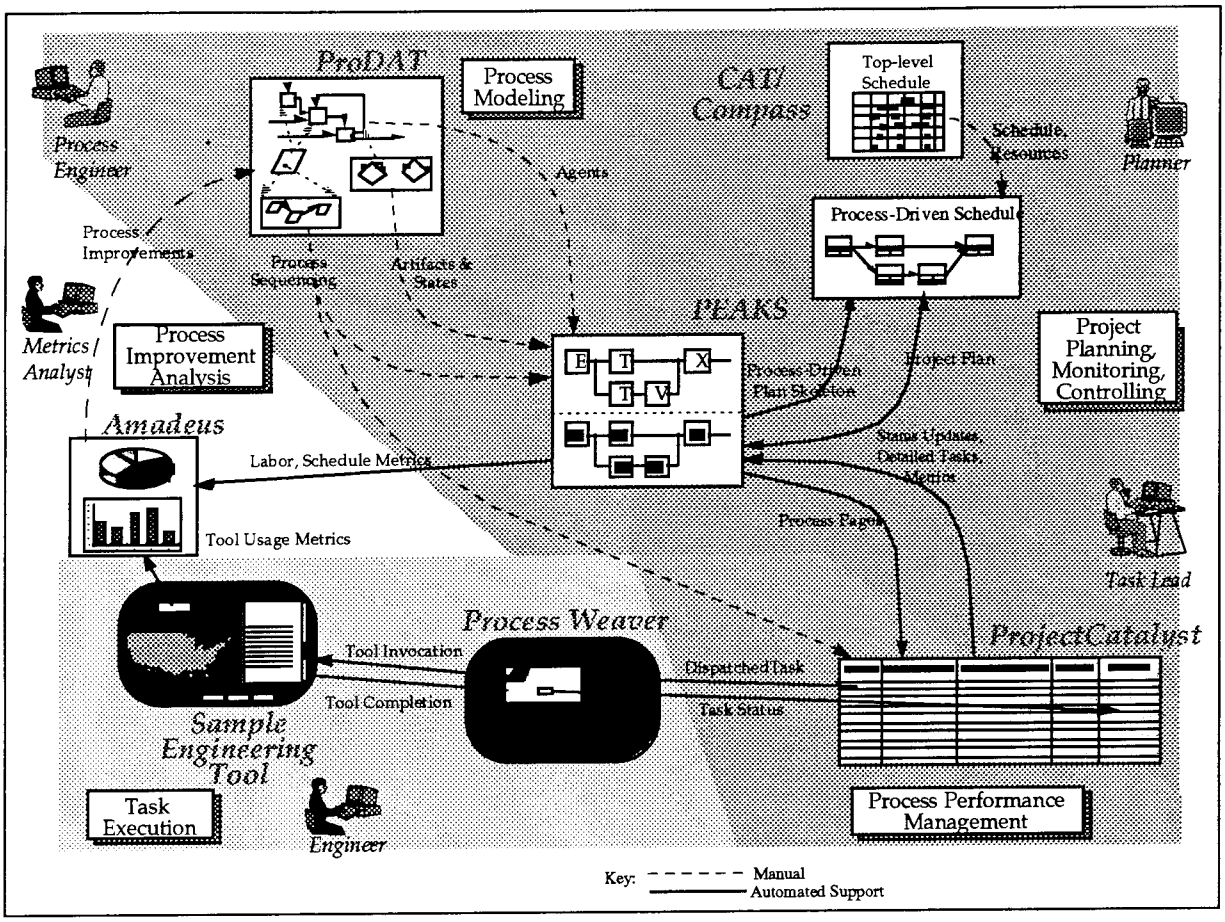


Figure 5. The Demonstration Project Process Support Environment

The PSE layer includes the STARS-sponsored tools PEAKS, ProjectCatalyst, and Amadeus. PEAKS is used to model the process using a graphical ETVX[1] notation and to construct a process-driven plan for the project. ProjectCatalyst [ProjCat94] is used to support the coordination and execution of the process-driven plan. To provide this support, ProjectCatalyst imports the PEAKS plan and presents it to task leads as a hierarchy of process component "pages". The pages list the work tasks, inputs, outputs, verification points, and lower-level process components called for by the process-driven plan. The task leads use these pages (which look much like spreadsheets) to dispatch tasks to engineers and to monitor status. They can also refine the pages to add additional steps not part of the formal process. As each task is dispatched to the designated engineer, ProjectCatalyst supplies a "work context" to Process Weaver, providing the engineer with convenient access to the artifacts (files, etc.) and the SEE tools to be used for the task. As tasks are completed and milestones are reached, status is reported dynamically to the PEAKS planning database -

[1] ETVX - Entry, Task, Validation, eXit - a notation for depicting process sequencing.

providing managers with up to the minute status of project activities. This closed-loop SEE mechanism assures the project that its work is taking place in accordance with the process and that status information is based on completion of prescribed validations. Further, as key points in the process execution are reached, measurement data (such as labor hours, elapsed clock time, automated quality measures, etc.) is captured in the Amadeus database, assuring that resulting metrics are similarly based on actual work processes - and allowing analysts to derive credible information to assist with process improvement.

The basic usage paradigm supported by the STARS PSE toolset seems promising to the end-users. Based on survey feedback, task leads seem at home with ProjectCatalyst's concise spreadsheet-like way of viewing their delegated work in progress and appreciate the automatic reporting of status based on the engineering work starts and stops. Engineers have indicated that they appreciate the structured way their tasks are maintained in Process Weaver agendas, as well as the ability to quickly navigate to their work product files from their work contexts.

As discussed in the Lessons Learned section, however, there are still implementation difficulties with the current tools that would have to be overcome before they could be considered for production use. As of this writing, while PEAKS is nearing commercialization, it appears that ProjectCatalyst will remain a prototype implementation. Current plans are to implement some of the ProjectCatalyst functionality in PEAKS and to continue to assess other avenues for supporting the automated enactment aspects of the PSE.

Although much work remains to fully realize a Process Support Environment, experience to date demonstrates the viability of the ideas. Interested readers should refer to the paper "Using Process to Integrate SEEs" [Randall95], also published in these proceedings - which further elaborates the motivation for the PSE layer, provides a model for its structure and interfaces, and discusses a number of lessons learned specific to integrating process and SEE.

To dramatize the relevance and timeliness of this work, the April, 1994 STSC Report on SEE Technology [STSC94] identified seven key SEE technology areas that have received insufficient attention to date and in which the industry now seems postured to make significant progress. Of these seven areas, the top three were process modeling, process definition, and process enactment and enforcement. The PSE encapsulation layer addresses all three of these areas.

REFERENCES

[Bristow95] Bristow D.J., Bulat B.G., Burton R. ***Product Line Process Development***, *Proceedings Seventh Annual Software Technology Conference*, Salt Lake City, UT, April 1995. (Published simultaneously with the present paper)

[Brown92] Brown, A. W., Earl A. N., and J. A. McDermid, ***Software Engineering Environments: Automated Support for Software Engineering.*** McGraw-Hill Book Co., UK, 1992.

[Bulat95] Bulat B.G. ***SWSC Domain Engineering Experiences***, *Proceedings Seventh Annual Software Technology Conference*, Salt Lake City, UT, April 1995. (Published simultaneously with the present paper)

[DemExp95] ***AF/STARS Demonstration Project Experience Report, Version 2.0 (Draft)***, CDRL Sequence A011-002D, Electronic Systems Center, AFMC, USAF, December 1994 (currently under review for comment by the Government).

[NGCR93] ***Reference Model for Project Support Environments, Version 2.0 (Draft)***, Next Generation Computer Resources, 2 September 1993.

[NIST93] ***NIST Special Publication 500-211, Reference Model for Frameworks of Software Engineering Environments*** (Technical Report ECMA TR/55, 3rd ed.), National Institute of Standards and Technology, August 1993.

[ProjCat94] ***ProjectCatalyst Users Manual***, Software Engineering Technology, Inc., September 1994 (available upon request from SET, 2770 Indian River Blvd., Vero Beach, FL 32960).

[STSC94] Hanrahan R., Daud C., Meiser, K., Peterson J. ***Software Engineering Environment Technology Report***, Software Technology Support Center, OO-ALC/TISE, Hill AFB, Utah, April 1994.

[SchMel92] Shlaer S., Mellor S.J. ***Object Lifecycles, Modeling the World in States***, Yourdon Press, 1992.

[ToolTalk94] ***Common Desktop Environment: Getting Started Using ToolTalk Messaging***, Sun Microsystems Inc., Mountain View, CA, 1994.

[Trimble94] Trimble J. (ed.), ***STARS Program History 1983-1993 Version 1.1***, available from the STARS Program Office.

Appendix H

Using Process to Integrate SEEs

The material in this section is intended to supplement Sections 3.0, Process Support, and 4.0, SEE Support. It consists of the complete text of a paper entitled "Using Process to Integrate Software Engineering Environments [RandallB95], to be presented at the Software Technology Conference, STC '95.

The paper discusses how process can be viewed as a vital asset in performing SEE integration. It presents a theoretical model for a Process Support Environment, which is a SEE abstraction layer that both supports all aspects of process engineering and execution and serves as a vehicle for SEE integration.

Presenters: Dr. Richard Randall, William Ett (Loral Federal Systems)
Title: Using Process to Integrate Software Engineering Environments
Track: 7 - Architectures
Day: Tuesday, 4/11/95, 10:30 AM
Keywords: SEE, Software Engineering Environments, Process, Integration, Architecture, Megaprogramming

USING PROCESS TO INTEGRATE SOFTWARE ENGINEERING ENVIRONMENTS

1. INTRODUCTION

Modern software engineering principles call for separating systems into encapsulation layers that can be specified, designed, and implemented largely independently from one another. Such layers help engineers:

- Gain intellectual control of a system by dividing it into meaningful abstractions,
- Reduce the complexity of building the system by dividing it into separable pieces that can be built independently by a relatively small number of specialists (or perhaps procured commercially),
- Maintain the system more economically by localizing the impact of most changes, and
- Evolve the system more easily by allowing replacement of whole layers of functionality.

A layered architecture is fundamental to the megaprogramming approach being followed by the Air Force/STARS Demonstration Project, as engineers develop the Space Command and Control Architectural Infrastructure application for NORAD and USSPACECOM. The layering strategy also applies to the software engineering environment (SEE) being used to support the application development.

This paper focuses on one particular SEE layer: a layer that leverages one of an organization's most valued assets - its process - and uses it as a basis for SEE integration. This layer, termed the Process Support Environment (PSE), conceived by Loral Federal Systems[1] on behalf of the STARS program, is being used on the Air Force/STARS Demonstration Project SEE, in conjunction with a major Air Force priority to establish and support a megaprogramming product-line process.

[1] *Formerly IBM Federal Systems Company*

The activity to produce and use the PSE has yielded useful lessons learned which can guide other organizations as they pursue their own process-driven SEEs. The experience also motivates future productization in this area: a customizable process overlay that can be added to orchestrate an organization's existing SEE. The potential exists to add process integration to existing SEEs with modest investment - largely consisting of well-defined instantiation procedures.

To dramatize the relevance and timeliness of these results, the April, 1994 STSC Report on SEE Technology [STSC94] identified seven key technology areas that have received insufficient attention to date and in which the industry now seems postured to make significant progress. Of these seven areas, the top three were process modeling, process definition, and process enactment and enforcement. The PSE encapsulation layer discussed in this paper addresses all three of these areas. The layer also provides a tailorable metrics instrumentation framework to help the organization assess and improve their process.

The remainder of the paper is organized as follows:

- Section 2, Context, provides background on the STARS Program and the Air Force/STARS Demonstration Project;
- Section 3, Process-Based SEE Integration, elaborates the notion of encapsulation layers, discusses its relevance to SEE integration, and offers a model of the PSE encapsulation layer;
- Section 4, The Air Force/STARS PSE: Experience To Date, describes how the Demonstration Project PSE is currently constructed and the extent to which it adheres to the PSE model provided in Section 3, and summarizes our experience to date in developing, transitioning, and improving the PSE; and
- Section 5, Conclusion, presents our priorities for continuing the evolution of both the implementation of the PSE and the underlying ideas.

The reader is invited to refer to a closely related paper also contained in these proceedings, "Integrating a SEE for Megaprogramming: Lessons Learned" [Randall95], which discusses the Demonstration Project's SEE integration experience from a broader perspective.

2. CONTEXT

2.1 THE STARS PROGRAM

The ARPA STARS program is a technology development, integration and transition program to demonstrate a process-driven, domain specific, reuse-based approach to software engineering that is supported by appropriate tool and environment technology - an approach referred to as "megaprogramming".

Megaprogramming

Megaprogramming is a product-line (family of systems) approach to the creation and maintenance of software intensive systems. It is characterized by the reuse of

software life-cycle assets within a product-line including common architecture, models and components. Megaprogramming also includes the definition and enactment of disciplined processes for the development of applications and the evolution of the product-line as a whole. Finally, megaprogramming calls for automated support for the process via advanced software engineering environment (SEE) capabilities and their integration among those tools. For a more in-depth introduction to the STARS program and the notions of megaprogramming, please refer to [Trimble94].

Demonstration Projects

The STARS program mission is to accelerate the transition to the megaprogramming paradigm. Key vehicles for bringing this about are the three STARS Demonstration Projects - one with each of the three services (Air Force, Army, and Navy) - which are currently engaged in applying the principles of megaprogramming to real systems. The major objectives for each of the projects are:

- Apply megaprogramming principles to the development of software for an actual DoD application, to establish the credibility of the approach.
- Collect and document experience about the benefits and costs of megaprogramming as well as the effectiveness of the specific tools and techniques used on the project, to help other organizations plan for and implement similar approaches.
- Transition to the Demonstration Project's parent organization, to establish the capability to apply megaprogramming to other applications in their product-line.

2.2 THE AIR FORCE/STARS DEMONSTRATION PROJECT

The Air Force's Demonstration Project was identified in 1992: the Space Command and Control Architectural Infrastructure (SCAI) project, to be managed by AFSPC's[2] Space and Warning Systems Center[3] (SWSC) at Peterson AFB, Colorado. Loral Federal Systems[4], one of three STARS prime contractors, was paired with the Air Force for the project.

The SWSC is responsible for the maintenance of the application software for the C^2 centers at the Cheyenne Mountain Air Force Station (CMAS) - which are responsible for national attack warning/assessment and space surveillance/defense/control. A large number of mission-critical systems are involved, with a high annual maintenance cost.

---

[2] *Air Force Space Command*

[3] *Effective Februrary, 1995, the SWSC is transfering to the Air Force Materiel Command (AFMC) and will be known as the Space and Warning Systems Directorate (SWSD)..*

[4] *Formerly IBM Federal Systems Company*

The SWSC, determined to apply new software technologies to reduce maintenance costs, had already been working to build up a megaprogramming capability; and the partnership with STARS was a natural way to accelerate the transition. Table 1 provides a summary of the progress to date on the project - in terms of the three megaprogramming technology thrusts being pursued by STARS.

	Original SWSC Posture	Accomplishments since Establishing STARS Partnership	Activities in Progress (as of 1/95)
Domain-Specific Reuse	• Strong architecture, based on RICC architectural infrastructure • Strong emphasis on Open Systems, commercial tools • Domain models underway • Commitment to Ada	• Demonstrated viability of architectural approach • Defined tailored specification standard based on Cleanroom, MIL-STD 498, and others • Completed object-based application models; specified SCAI system and first two SCAI releases • Developed and tested SCAI Release 1	• Developing SCAI Releases 2; specifying Release 3 • Refining product-line architectural framework • Continuing to develop domain models
Systematic Process	• Understanding of importance of process • SWSC Software Engineering Process Group (SEPG) established • Semi-formal process definition in selected areas • Corporate Information Management (CIM) IDEF model underway	• Instituted formal approach to process definition, based on STARS/SEI collaboration • Created a product-line process architecture • Integrated OO, Cleanroom, and the Ada Process Model methods • Formally defined processes for Application Engineering (AE) • Launched a major metrics initiative • Automated staff hour and defect metrics collection	• Nearing completion of formal definition of CM process • Beginning formal definition of Domain Engineering process • Working on second round of AE specification process • Using automated process modeling and enactment support for SCAI Release 2 and 3
Automated Support	• Commitment to Rational Ada support product-line • Commitment to Universal Network Architecture Services (UNAS), and Reusable Integrated Command Center (RICC) for Architectural Infrastructure	• Integrated a state-of-the-art open systems SEE: IBM and Sun platforms, Rational and TRW toolsets • Installed advanced process support toolset; encoded and began automated enactment of SCAI Release 2 and 3 processes • Instituted automated tracking capability for problems, action items, etc. • Began use of Rational SoDA for automated document production	• Enhancing the functionality and integration of the process tools • Applying Amadeus to automate collection of SEE usage metrics • Extending process automation across geographical locations

Table 1. Air Force Demonstration Project - Megaprogramming Progress and Status

2.3 Demonstration Project SEE

The Demonstration Project SEE is composed of approximately 50 workstations connected to 3 server-class machines. The network is distributed across two geographical sites (over a high-speed link) and is about evenly divided between Sun and IBM Unix-based platforms.

The SEE is populated with a set of tools to support the desired end-user functionality, as depicted in Figure 1. The SWSC has committed to the use of Ada for the application product-line, and has selected the Rational toolset (Apex, Verdix, RCI, Ada Analyzer, SoDA, and Rose) as a key part of the SEE. In addition, because the application architectural strategy is based on a TRW-originated architectural infrastructure, the SWSC has adopted the corresponding toolset (SALE, RICC Tools) as another key part of the SEE. The diagram also discloses the organization's emphasis on process technology (the Process Modeling, Project Management, Process Enactment, and Metrics functionality clusters).

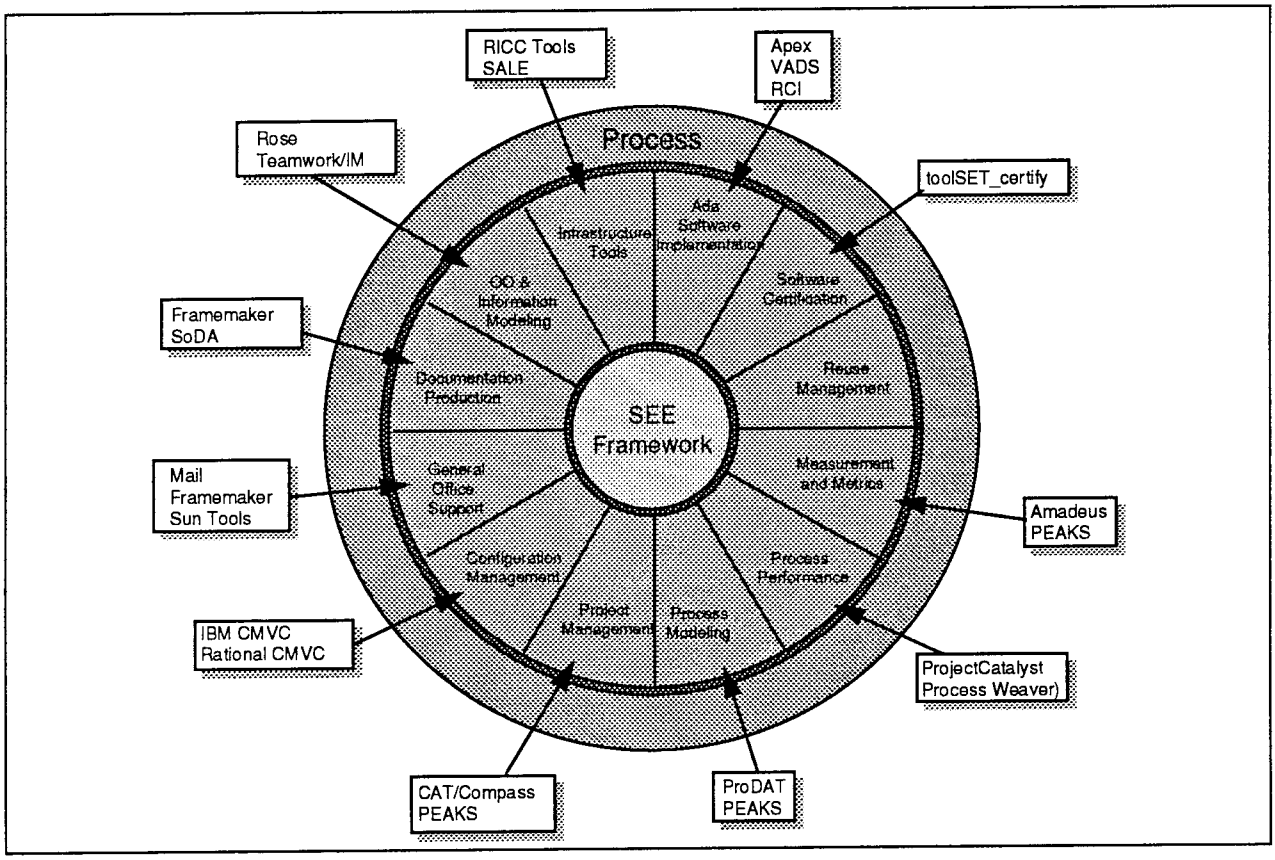


Figure 1. Demonstration Project SEE: Tool Functionality Groups

Table 2 identifies the supplier of each tool shown in Figure 1. As shown, the SWSC has selected Rational and TRW as major toolset providers for the Demonstration Project. Also shown in the table are four tools that have been developed with STARS support.

Type	Tool Name	Vendor/Developer	Purpose
Rational Ada Development Toolset			
	Apex	Rational	Code creation and testing
	RCI	Rational	Interfaces Apex with other Ada compilers
	Rose	Rational	Object oriented analysis and design
	SoDA	Rational	Automated document generation
	VADS	Rational	Verdix compiler
TRW Architectural Infrastructure Support Toolset			
	RICC Tools	TRW	Application display, message, and database definition
	SALE	TRW	Application network definition (used with UNAS)
	UNAS (Universal Network Architecture System)	TRW	Application network manager
STARS-Supported Tools			
	Amadeus	Amadeus Software Research, Inc.	Metrics repository and analysis
	PEAKS	Cedar Creek Process Engineering (ccPE)	Process modeling, planning, and plan simulation
	ProjectCatalyst	Software Engineering Technology (SET)	Low-level process definition and process execution
	toolSET_certify	SET	Cleanroom certification testing support
Other Tools			
	CAT/Compass	Robbins-Gioia	Project management
	CMVC	IBM	Configuration management and tracking system
	FrameMaker	Frame Technology, Inc.	Documentation and publication
	ProDAT	Embedded Computer Resource Support Improvement Program (ESIP), managed by Sacramento/ALC	IDEF-oriented process definition
	Process Weaver	Cap Gemini America	Process workflow manager
	SunTools	Sun	General office support
	Teamwork/IM	Cadre Technologies	Information modeling

Table 2. Demonstration Project SEE Tool Suppliers

The integration of the SEE tools is accomplished through a combination of techniques and mechanisms. Control integration (tool invocation and communication) is provided by

- IBM AIX SDE WorkBench/6000 broadcast messaging service, the Process Weaver message service, operating system process services, and TCP/IP sockets. The WorkBench and operating system process control provided local service within a user's machine session. Process Weaver and TCP/IP provided service between users and between machines.
- Data integration is provided by the Oracle Relational Database Management System (RDBMS) and the operating system file system.
- Presentation and user interface integration is provided by the X Window System and the Motif window manager.
- Process integration is provided by the STARS-sponsored Process Support Environment toolset: PEAKS, ProjectCatalyst (in conjunction with Process Weaver).

3. Process-Based SEE Integration

This section elaborates the notion of encapsulation layers, discusses their relevance to SEE integration, and offers a generic model of the Process Support Environment (PSE) encapsulation layer.

3.1 Encapsulation Layering and SEE Integration

The Importance of Encapsulation Layers in Software System Architecture in General

The importance of encapsulation layers is emphasized in [SchMel92][5]. To illustrate the notion of layering, consider a system for with a human-machine interface (HMI) layer which encapsulates the specifics of the operator's physical interface with the system (i.e., the display and console interaction). This example can be used to identify some of the hallmarks of encapsulation layers:

- The essential requirements for the layer are the rules about the system's usage paradigm plus the application programming interface needed by the system's higher levels
- The layer is expected to provide an implementation that binds to lower-level details - in this case, device and operating system characteristics

The power of such encapsulation stems from these characteristics:

- It allows reasoning about the capabilities of the layer with minimal concern about other aspects of the system

[5] *Schlaer and Mellor use the term "domain" to refer to what we are calling "encapsulation layer".*

- It allows maximum efficiency in applying experts in the layer's domain (in this case display interfaces)
- It facilitates replacing the layer's implementation (e.g. using new technologies) with minimal impact to other parts of the system

Architectural Layering in The Demonstration Project Application

On the Air Force/STARS Demonstration Project, architectural layering is viewed as essential to laying the foundation for a megaprogramming product line. Figure 2 provides a high-level depiction of the architecture being used for the SCAI application. Note that three major encapsulation layers provide the foundation for all applications in the future product-line. The lower two layers are off-the-shelf, and the third layer (architectural infrastructure) provides product-line specific encapsulations for such services as User Interface, Data Management, and Message Handling. Interested readers are invited to refer to a more detailed discussion of the SCAI architecture found in [Bristow95] and [Bulat95].

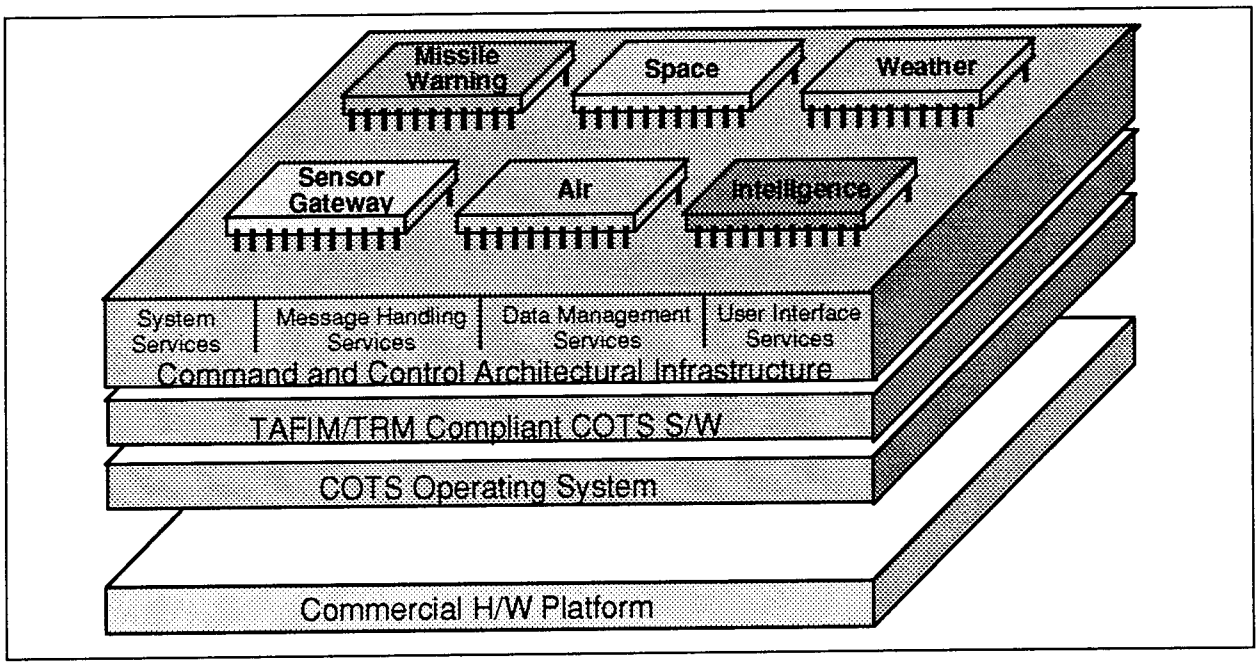


Figure 2. Layered Architecture for Demonstration Project Application

Architectural Layering Applied to the SEE

We now move on to discuss how the above general notions about encapsulation layers apply to SEE integration. Table 3 is taken from the current draft of the Air Force/STARS Demonstration Project Experience Report [DemExp95]. It analyzes several architectural characteristics of the Demo Project SEE. Perhaps the most important characteristic is the layered architecture: as discussed above, implementation of each layer can proceed with minimal concern for the specifics of the implementation of the other layers.

Major Architectural Characteristics	Example Sub-Categories	SCAI SEE Domain Implementation Examples
• Layered architecture	• Common underlying operating system environment • Process support environment • Major functionality encapsulations	• Unix, TCP/IP, NFS, X • STARS PSE toolset • Rational's integration family of Ada support tools (centered around Apex) • TRW's integration family of code generation tools supporting the architectural infrastructure (UNAS, RICC)
• Common user interface	• Common window-handling • Common window behavior look and feel	• Motif • Open Interface (from Neuron Data), Display Builder (from TRW)
• Common program interface mechanisms	• Low-level API services • High-level data repository services	• Broadcast Message System (part of HP's SoftBench) • TCP/IP Sockets (for guaranteed message delivery) • COTS DBMSs (Oracle, Sybase)
• Component reuse	• COTS tools	• Various

Table 3. Architecture Characteristics for the SEE Domain

Figure 3 provides a view of SEE architecture in terms of some typical encapsulation layers, including the Process Support Environment layer which is the focus of this paper. The arrows show dependence of one layer on another (i.e., we are not showing data flow or control flow here).

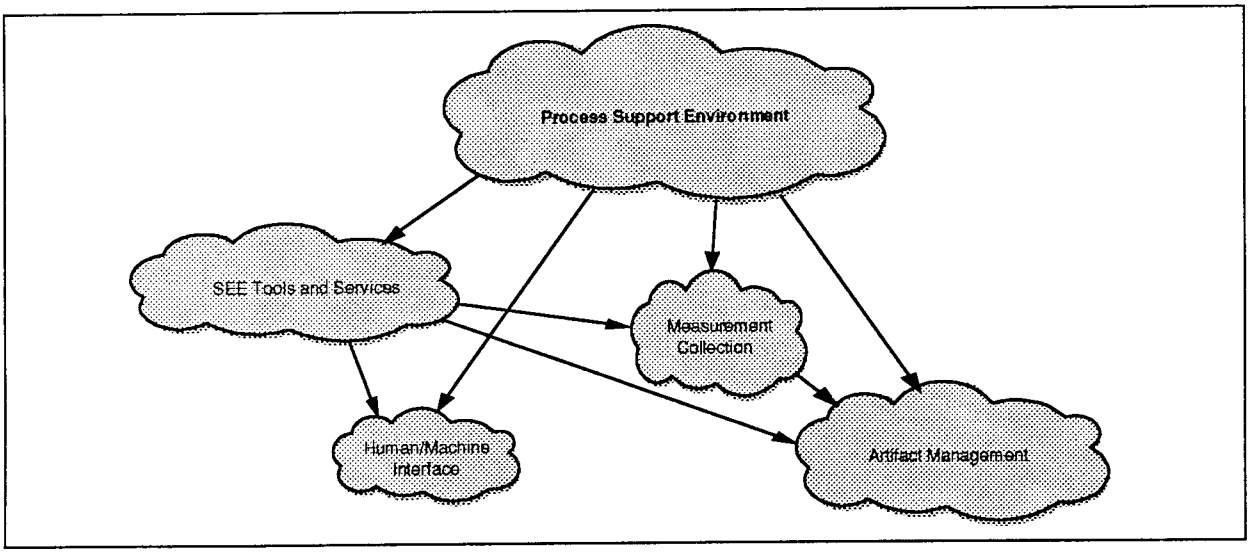


Figure 3. SEE Encapsulation Layers

Encapsulation Layers: A Fundamental SEE Integration Strategy

Just as with other software systems, a SEE integrated using well-engineered encapsulation layers will exhibit more effective functionality and will be more maintainable and evolvable, since:

- Integration within encapsulation layers can take advantage of the knowledge and expertise of a relatively small number of experts, where experience has shown that tight integration is the most successful.
- Less integration will be needed between distinct encapsulation areas (simplifying maintenance), and can be restricted to standardized interfaces (facilitating replacing a layer with an improved alternative).

Implementing SEE Encapsulation Layers using SEE Integration Mechanisms

It is important to distinguish "encapsulation layers" from another term that is commonly used in discussions of SEE integration: "integration mechanisms". An encapsulation layer is an encapsulation of functionality, while an integration mechanism is a means of joining components for the purpose of combining functionality. The components within a layer may be joined together via integration mechanisms - and, further, the layer as a whole may be joined with other layers via the same mechanisms.

To implement these encapsulation layers and to interconnect them, integration mechanisms are used. Here are the most commonly cited integration mechanisms[6], with brief examples for how each is used:

- Control integration mechanisms permit the invocation of a service provided by another component (perhaps including the launching of an entire application). Such mechanisms include direct calls, sockets, mailboxes, and broadcast messaging. Examples of the facilities providing the latter type of services are Hewlett-Packard's SoftBench[7] Broadcast Message Service (BMS) and Sun's ToolTalk[8].
- Presentation integration mechanisms permit the development of application wrappers that provide users with a uniform interface for the invocation of SEE applications and services. An excellent example of a presentation integration service is the one found in HP SoftBench.
- Data integration mechanisms permit application developers to share common data objects across applications, where each application developer uses a database or object manager, making the API for these applications known to other application developers. Data integration between heterogeneous software application vendors has had limited success.

---

[6] See, e.g, the August 1993 NIST/ECMA reference model for SEE frameworks [NIST93]

[7] HP SoftBench is a registered trademark of the Hewlett-Packard Corporation.

[8] ToolTalk is a registered trademark of Sun Microsystems.

- Process integration mechanisms support the coordinated use of the SEE's other integration mechanisms (control integration, data integration and presentation integration mechanisms) to support the execution of work steps within an automated process sequence (or task).

  Figure 4 illustrates the use of a process integration mechanism to bind a conceptual representation of a process segment, shown at the top of the figure, to specific SEE services and data - using control, data and presentation integration services.

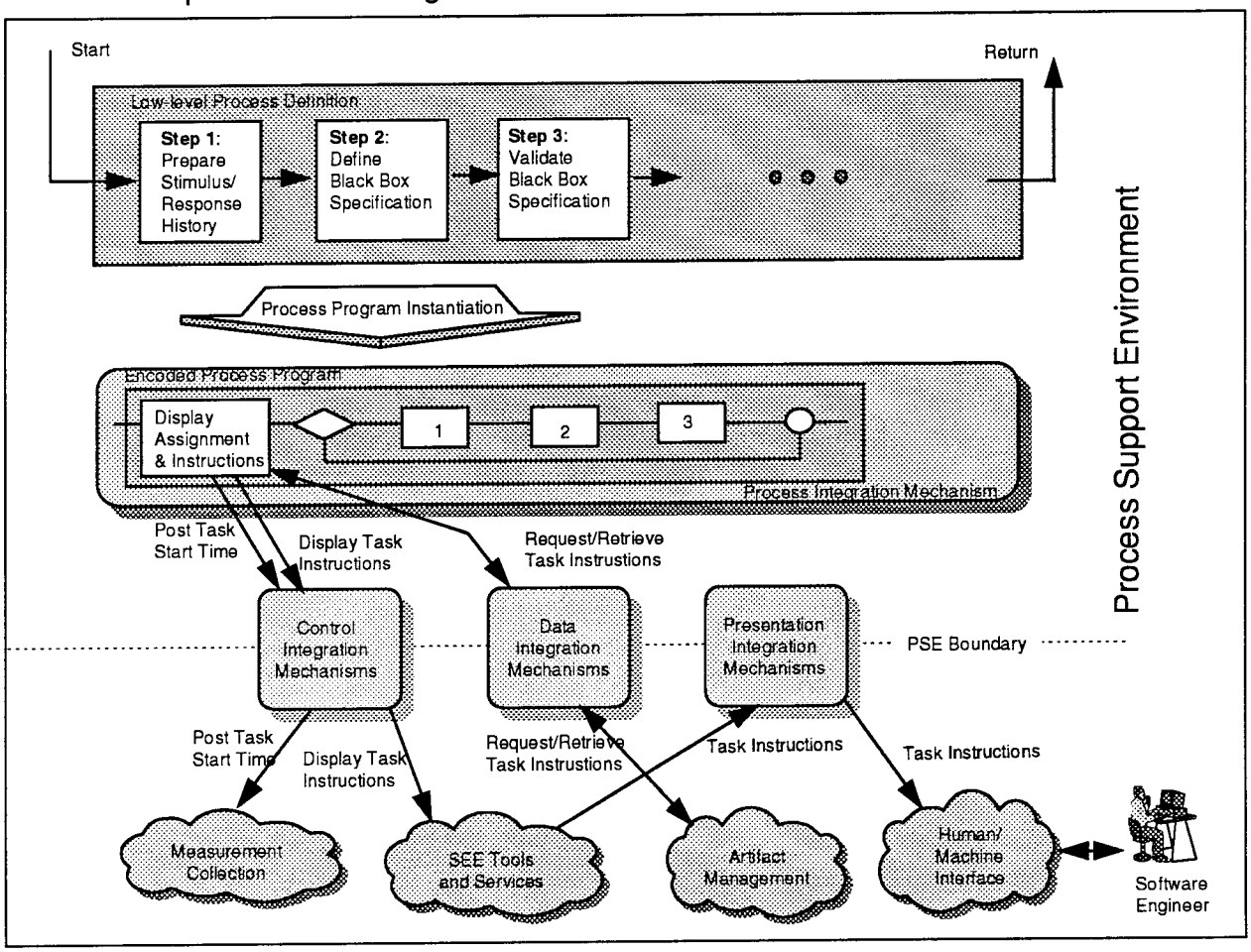


Figure 4. Supporting a Task via Process Integration (1 of 2)

The conceptual view is translated into a process program view, shown in the middle of the figure, in which process engineers have arranged for a new step to be executed: "Display Assignment of Instructions". This new step, which only makes sense when the process segment is supported by the PSE, posts the task start time to a measurement database and causes detailed process enactment description information to be automatically displayed to the practitioner.

This process program is executed by a "workflow engine", which uses an encoded representation of process sequencing to progress through the

steps. Associated with such an engine are process programming facilities that allow process engineers to bind each step to SEE artifacts and tools. The workflow engine thus provides the facilities to implement process integration.

As shown in Figure 4, the execution of the first step involves the use of the three other integration mechanisms to record the prescribed measurement data and to display the process instruction data.

Figure 5 is a continuation of this same example, showing how a later step, "Define Black Box Specification" is supported.

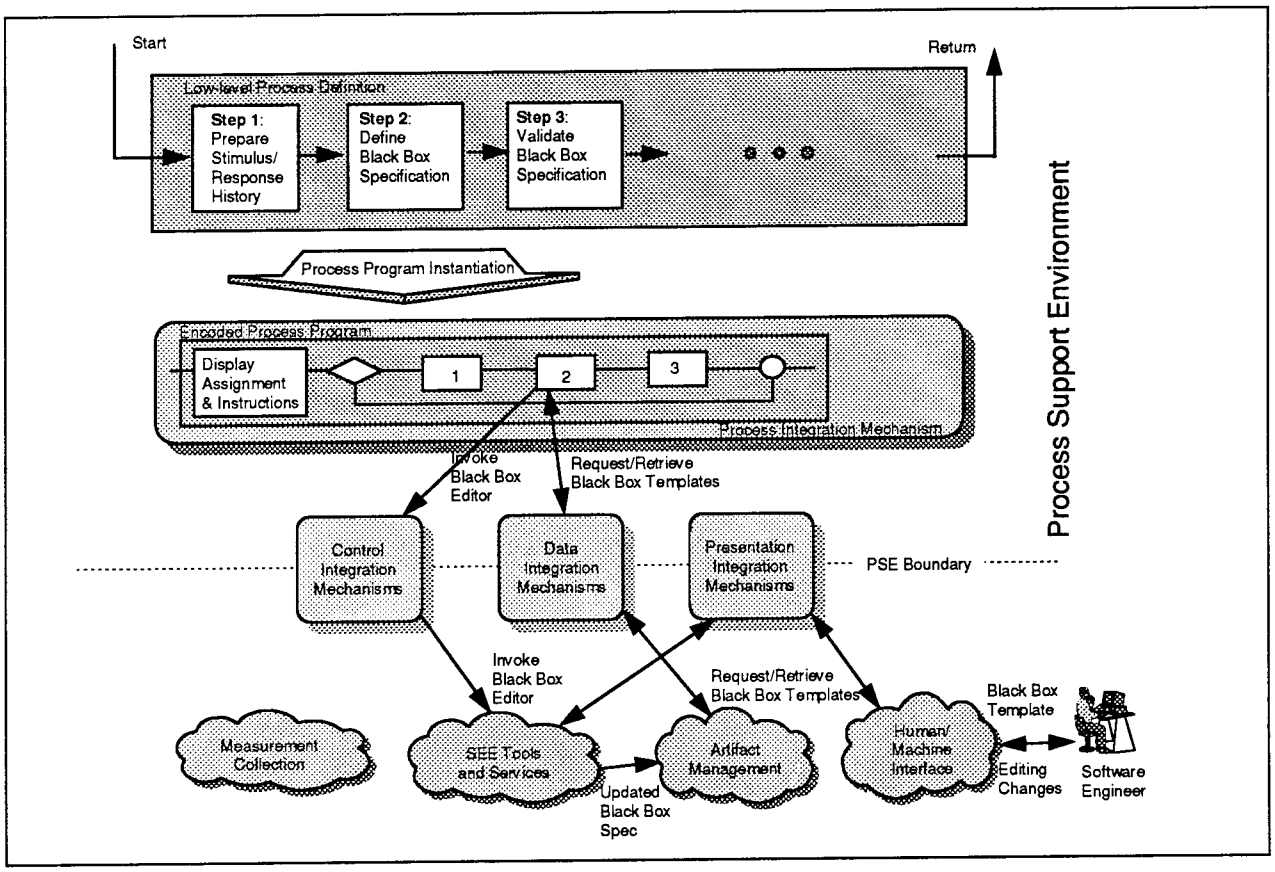


Figure 5. Supporting a Task via Process Integration (2 of 2)

This step is implemented by using a Control Integration Mechanism to invoke an editor, and a Data Integration Mechanism to retrieve the project's standard template for black boxes.

Examples of commercial products that provide process integration services - also referred to as workflow automation services - are HP Synervision[9], Process Weaver[10], FlowMark[11] and InConcert[12].

[9]HP Synervision is a registered trademark of the Hewlett-Packard Corporation.

[10]Process Weaver is a registered trademark of Cap Gemini Innovation.

Please refer to Section 4.3, Lessons Learned, starting on page 23, for a discussion of some of our experiences in applying SEE integration mechanisms.

3.2 The Process Support Environment Encapsulation Layer

The foregoing discussion has developed the importance of encapsulation layers to SEE integration. The remainder of this paper focuses on one particular encapsulation layer: the Process Support Environment (PSE) layer. The authors' contention is that this layer is one of the most vital SEE integration priorities, because it directly joins the organization's process to the SEE that supports it. As will be shown, use of a PSE can not only tie the SEE's functionality together for the end-user, it can also keep the process alive - and improving - by making it part of every user's routine use of the SEE. This dual integration strategy is illustrated in Figure 6.

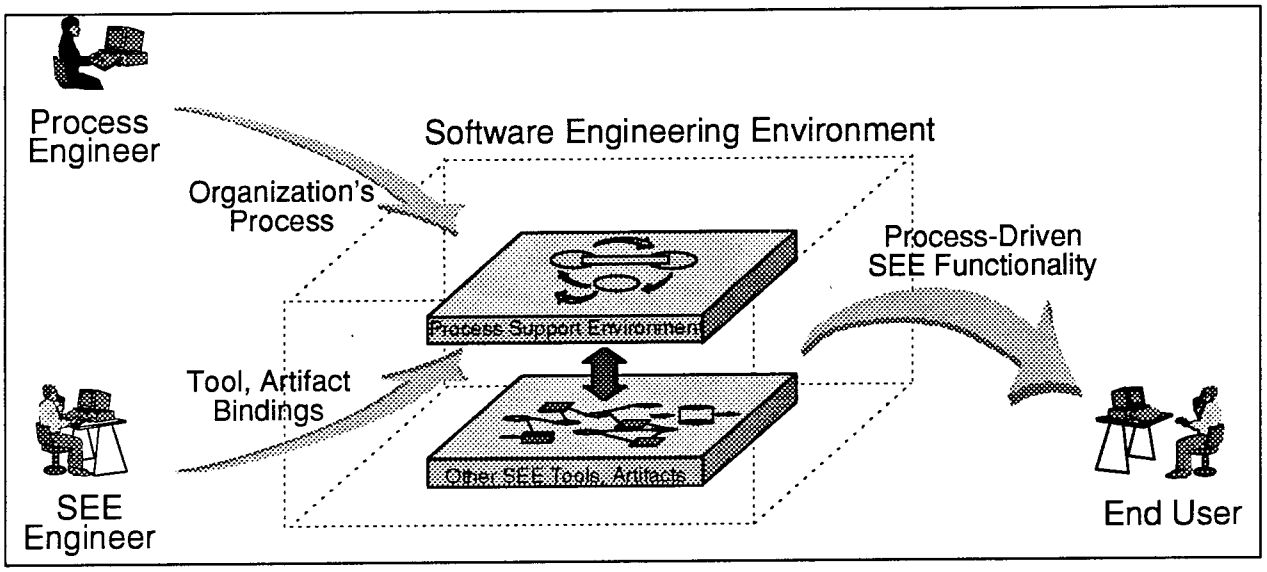


Figure 6. Process Support Environment Encapsulation Layer

Conceptually, instantiation of the process integration layer proceeds via a "lamination" procedure: the process and metrics requirements are added to the top, and the tool interface provisions are added to the bottom. The result is an encapsulated SEE integration layer that joins process to SEE.

A list of criteria for recognizing encapsulation layers was provided on page 7. As shown by the following analysis, the PSE encapsulation layer seems to fit the mold:

- The essential requirements are provided by the process, since the objective of the SEE is to support the organization in carrying out its process

---

[11] *FlowMark is a registered trademark of the IBM Corporation.*

[12] *InConcert is a registered trademark of the Xerox Corporation.*

- The "API" with the system's "higher levels" are the user interfaces:
  - For process engineers, to define and model the process
  - For managers, to plan a project based on the process
  - For practitioners, to carry out their work in the context of the process
  - For all users (including the above), to assess how well the process is working and to pursue high-payoff avenues for process improvement
- The organization can reason about the process integration layer with minimal concern for the specifics of the rest of the SEE
- The implementation of the layer is performed by process technology experts that are intimately familiar with state-of-the-art process thinking, available process tools, and SEE integration techniques
- Customization provisions allow organizations to modify other aspects of the SEE - such as changing compilers or text processing toolsets - with minimal perturbation to the layer or its users.

3.3 A Look Inside the Process Support Environment

This subsection provides a model of how such a Process Support Environment (PSE) might be constructed. The model will provide the basis for our discussion of the Demonstration Project PSE, and it will also assist in comparing and contrasting our approach with that of others.

3.3.1 PSE Requirements

Our view of the PSE's structure is motivated by the following general requirements.

The PSE should:

- Provide process automation options ranging from guidance to complete automation; the degree of process automation is always the organization's prerogative.
- Allow incremental process definition, implementation, and improvement.
- Support process definition via graphical modeling as well as conventional documentation. Preferably, the modeling syntax should be geared to describing process and the semantics should permit consistency checking.
- Support "process-driven project management". By this we mean:

- Allow project plans to be developed and evolved in the context of the organization's process. Allow project managers to identify the process basis for planned project activities.
- Allow management control to be tied to the process-driven plan. Managers and task leads should have the ability to discern the process context of the activities they are coordinating.
- Allow status monitoring to be automatically tied to process execution. Managers and task leads should be assured that milestones are reached via the defined process - including the successful completion of any predefined validations.

We regard management support capabilities to be part of the PSE because of the intimate relationship between process and management.

- Support automated assistance of process execution, providing process guidance, facilitating artifact navigation and tool usage, and enforcing conformance to process-defined standards in as unobtrusive a fashion as possible.
- Support measurement capture during the course of process execution, automatically where possible.
- Support metrics analysis, based on the above process-driven measurements as well as other means (e.g., analysis of source code), as a key ingredient of process improvement and technology transition.

The current Demonstration Project PSE partially satisfies all of the above requirements.

3.3.2 PSE Structure

Figure 7 provides a abstract view of the PSE (the shaded regions of the figure), showing its internal structure and its relationship to the rest of the SEE. The figure identifies several distinct capability sets within the PSE:

- Process modeling
- Project planning, monitoring and control
- Process performance management
- Task execution
- Process improvement analysis

Figure 7 also shows the PSE layer using services in the other encapsulation layers shown on Figure 3 on page 9, namely:

- SEE tools and services (e.g., compilers, object-oriented modeling tools, etc.)

- Measurement collection services for posting, storing and retrieving measurement data
- Artifact management services for accessing and storing artifacts and information about artifacts - including configuration management.
- Human/machine interface services (not explicitly shown in the figure).

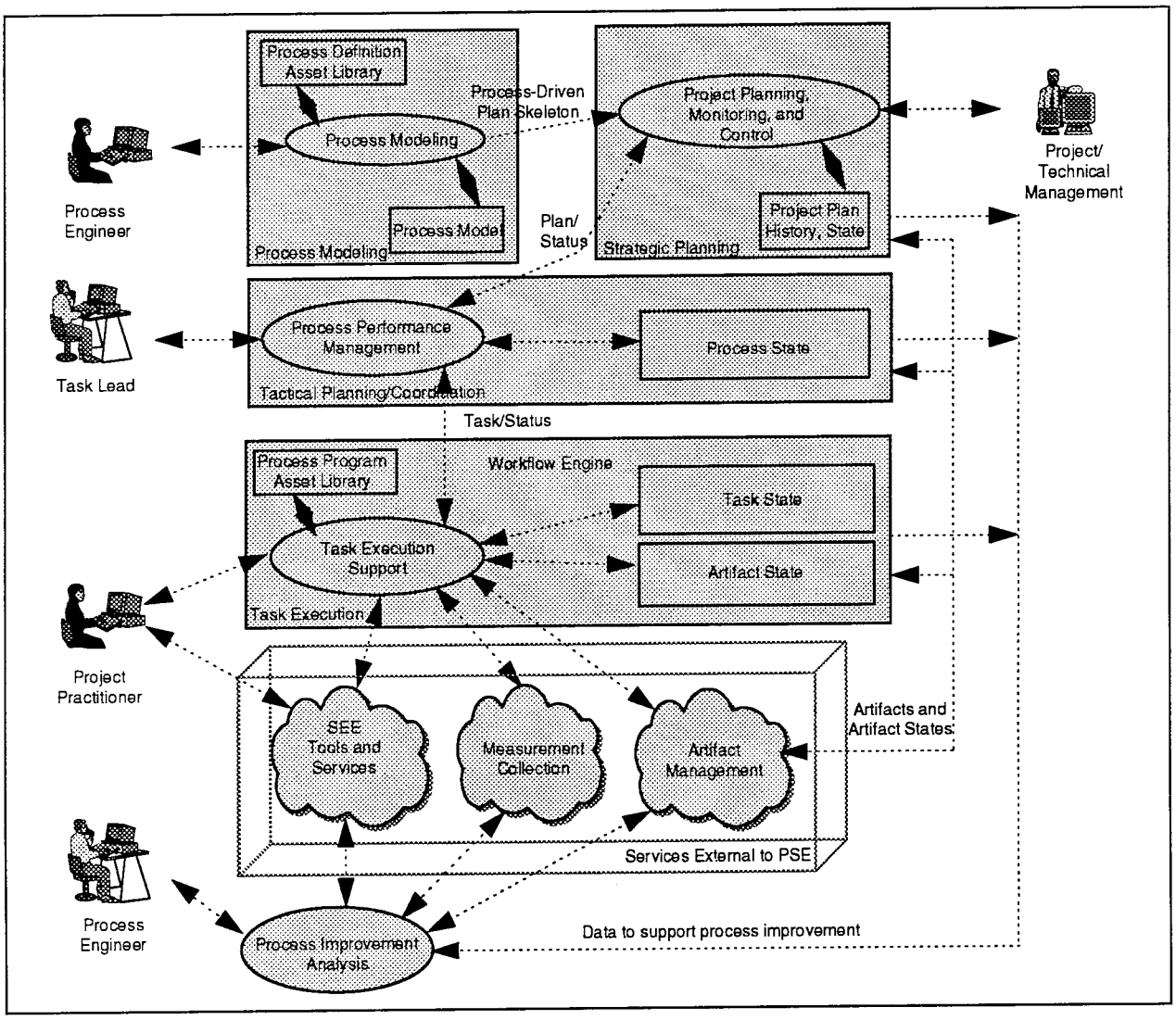


Figure 7. Process Support Environment: Structure and Interfaces

3.3.3 Description of PSE Capability Sets

Process Modeling (PM) Capability Set

This set provides process engineers with the tools necessary to define the organization's process, comprising:

- An architecture for the process, describing all of its components and their relationships
- The flow of artifacts within a process and its components (data flow)
- A constrained process activity network suitable for supporting project planning
- The control flow and control structure of activities within a process and its components (process/process component control structure)
- The composition and description of the artifacts involved in the process
- The quality characteristics established for each artifact the project is to produce and the completion criteria each artifact must satisfy
- The resource types required to support each identified activity
- A description of each process component, such that each process component has sufficient information available for its user to manually perform it.

Process specifications that are defined and tailored using the PSE's process modeling capabilities are maintained in a process asset library for use by current and future projects.

The process specifications prepared and adopted to support a project, become a key ingredient in the planning, design and implementation of process programs. A process program is a program designed to provide assistance to SEE users in following a defined process, as they perform their project work.

Project Planning, Monitoring and Control (PPMC) Capability Set

This set supports project and technical managers working with process engineers to initially plan software development projects. Further, once these projects are initiated, this capability set provides support to project and technical managers who monitor and control the project.

The PPMC Capability Set provides support for planning software development projects by providing the following capabilities:

- Automatic project plan instantiation from process specifications that provide the constrained activity flow for the project
- Analysis of proposed project plans for cost and schedule reasonableness.

The PPMC Capability Set provides support for process component delegation to technical management and their task leaders to address how they plan to satisfy the objectives of a delegated process.

Once a project is initiated and processes have been delegated for detailed planning and management, the PPMC Capability Set provides support for monitoring and controlling the project though the following capabilities:

- Automatic posting of project status and events for weekly, bi-monthly and monthly project status report generation and analysis
- Programmable project exception reporting on selected project events, e.g. schedule anomalies, cost anomalies, resource anomalies, etc.
- Support for project replanning, upon process conditions requiring the project schedule to be revised, e.g. review failures, earned value problems, etc.

Project activity and process state information is kept in a persistent object store for use in supporting report generation and automated event recognition.

Process Performance Management (PPM) Capability Set

This set supports technical managers and task leaders who plan and manage the detailed tasking necessary to address the specific requirements of a delegated process. As processes are not intended to dictate how technical work will be performed, technical managers and task leaders must plan the tasks they feel are necessary to achieve expected results and to coordinate these activities with their technical team.

The PPM Capability Set provides a process/task planning and management spreadsheet for use in defining and coordinating lower level processes and the tasks necessary to satisfy them. This spreadsheet provides the following capabilities:

- Low level process definition (processes below those of the ones defined in the project plan)
- Task planning to support the requirements of a delegated process
- Process delegation and tracking
- Task assignment, dispatching and tracking.

The process/task planning and management spreadsheet displays an overview of every process task, and as work proceeds and project milestones and process events occur, the spreadsheet is updated to reflect these changes. Detailed information is summarized for automatic reporting to support the PPMC Capability Set.

Project task state information is kept in a persistent object store to support the summarization of detailed data for use in supporting report generation and automated event recognition. Project artifact state information is also kept in a persistent object store to provide information for the PPM Capability Set about artifacts that exist and their usage, completeness and version status.

It is at the PPM Capability Set level that requirements for designing and implementing process programs must be made. Process programs may either be custom crafted for each process in the project's process architecture, or generic process programs may be prepared to support activity classes. The PPM Capability is responsible for dispatching tasks for project personnel to perform. This task

dispatching involves the invocation of a process program to support project personnel in performing their assigned task.

Task Execution Support (TE) Capability Set

This set supports project personnel in performing their work tasks, and provides task status information to the PPM Capability Set to support its mission of task tracking. It includes a workflow engine programs (process integration mechanism) for executing low-level process programs. The engine uses other SEE integration mechanisms (e.g., control, data, and presentation mechanisms) to interface with services external to the PSE layer. For example, a process program may automatically invoke a SEE tool, and a few steps later, request a measurement engine agent to post a process measurement.

The TE Capability Set may be implemented according to one or more automation paradigms:

- Passive support for project personnel, through the use of electronic mail and a hypertext description of the process currently being performed.
- Complete automation, where shell scripts (or canned procedures) are launched based on a SEE or process event.
- Proactive work flow management, involving both a human and the SEE. In this instance, work is assisted by an engine that provides convenient access to the appropriate tools and data - and provides guidance to support the practitioner's performance of task steps.

Since the latter automation paradigm is the most general capability (a coordinated mix of human and SEE activities) we will assume PSE capabilities include a proactive work flow execution support, employing a work flow engine, such as Process Weaver or InConcert.

When a task is dispatched by the PPM Capability, a process program to support that task begins its execution. Each process program includes a set of work steps that a task performer must accomplish. These work steps are made available to the task performer, based on the process program's control structure. Thus the process program, using the work flow engine, manages a dispatched task through to its completion. Process programs may be instrumented to provide automated measurements and validation steps:

- Task start and completion times can be automatically recorded, as well as task accumulated elapsed and environment usage time. This instrumentation is typically done by interfacing with a measurement engine, where measurement collection agents are invoked to collect and store the require measurement.
- Product reviews can be facilitated, by prompting with prescribed questions about the quality of an artifact that an engineer produced.

To increase the level of support, TE can provide automated launching of tools against the key artifacts being consumed and produced by the task. To accommodate this, a binding strategy must be implemented within the PSE, as indicated in Figure 8. Such binding can allow TE to simplify the practitioner's work by eliminating the need to navigate to the proper artifact and invoke the appropriate tool for it.

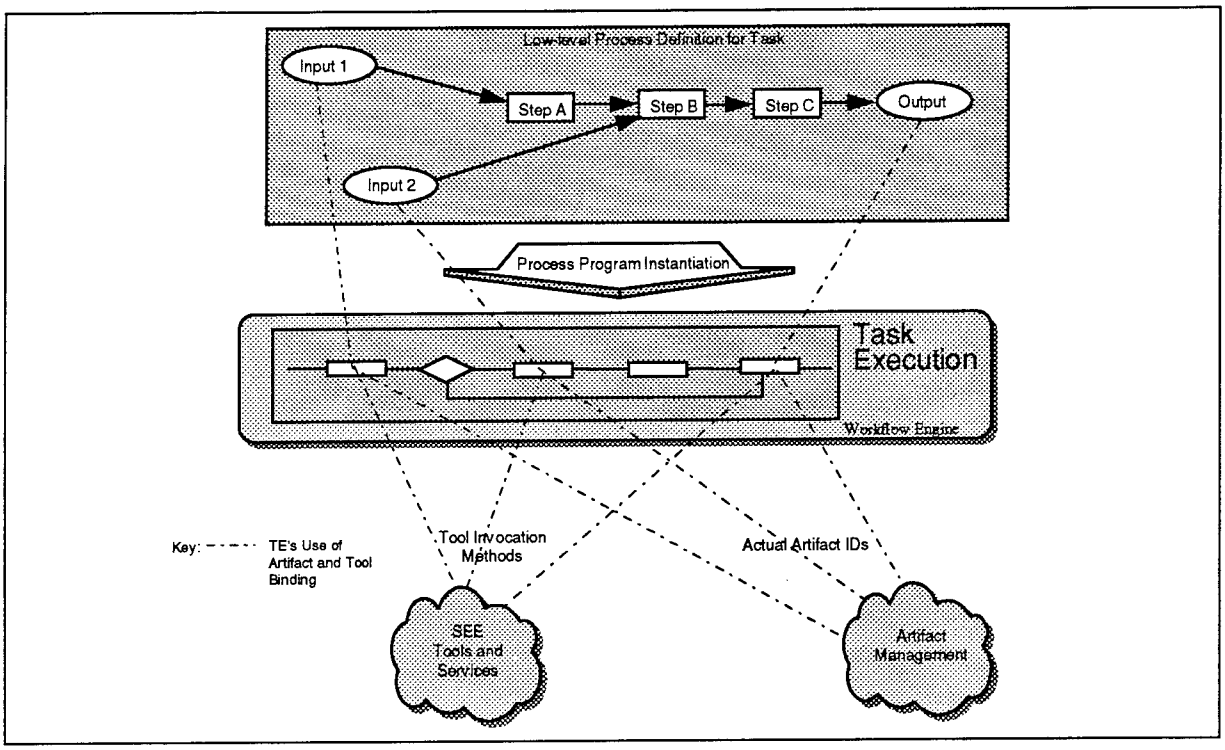


Figure 8. Binding Process Programs to SEE Artifacts and Tools

Global and local task state information is maintained in a persistent object store for the work flow engine to support task execution, intertask synchronization, task redirection, or task suspension. TE also maintains artifact state information to support task execution, intertask synchronization and task suspension.

Process Improvement Analysis (PIA) Capability Set

This set provides process engineers with access to the tools and resources to support both quantitative and qualitative analysis of process performance data and product quality data. Process engineers employ many techniques to support process improvement from reviewing the comments of process practitioners (project personnel who follow processes) to statistical analysis of data collected by the PSE Capabilities and the Measurement Collection Capabilities. Process engineers may employ statistical analysis tools to analyze process performance and project quality data to analyze performance times and defect ratios against existing norms. Further, using measurement data collected during process execution, metrics may be computed to support process improvement analyses. Typical PIA Capabilities include:

- Statistical analysis support for trend analysis and curve fitting

- Control chart preparation and analysis
- Problem cause and effect analysis.

3.4 Related Work

We have provided an appendix to the paper, "History of PSE Evolution", starting on page 34, to provide additional background to interested readers. The appendix discusses three antecedent PSEs developed under the Loral Federal Systems STARS contract, as well as a contrasting PSE under development on the Arcadia project. The PSEs are all analyzed in the context of the model of PSE structure and interfaces illustrated in Figure 7 on page 16.

4. The Air Force/STARS PSE: Experience To Date

4.1 PSE Description

Figure 9 illustrates the current Demonstration Project PSE. The diagram shows how the major PSE model components work together to support the process definition/-planning/performance/improvement cycle.

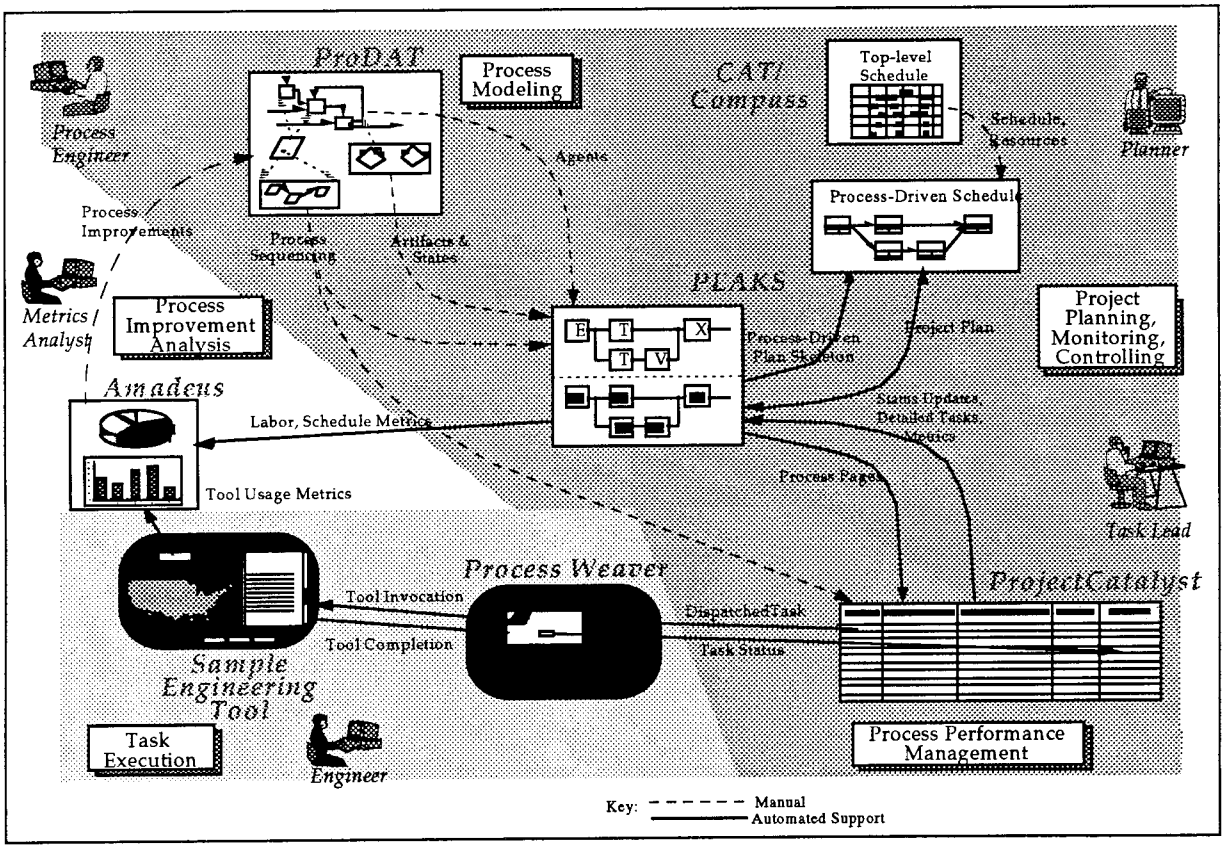


Figure 9. An Operational View of The Demonstration Project PSE

Process definition and modeling are supported by ProDAT, for activity modeling, and by the STARS-sponsored Process Engineering and Kernel System (PEAKS[13]), for work flow. These tools are being used by the Demonstration Project team to capture and analyze process in a form consistent with the STARS/SEI Process Definition Information Organizer Templates.

Process-driven project management is supported by PEAKS, which builds project plans directly from the process and allows management to apply resources and schedule constraints - as well as to simulate the impact of process modifications and quality failure probabilities.

Process enactment is supported by the STARS-sponsored ProjectCatalyst, and Cap Gemini's Process Weaver. ProjectCatalyst receives enactment specifics from PEAKS and allows task leads to manage the team's activities to follow the process-driven plan. Based on the project's customization information, ProjectCatalyst automatically instantiates Process Weaver process used by the practitioners to launch the appropriate tools for creating and manipulating the desired artifacts. As work actions are taken (starts and completions), ProjectCatalyst reports task completion date and effort to PEAKS, so that management can always obtain an accurate data concerning task and project status.

Measurement collection and metric computation is supported by both PEAKS and Amadeus. PEAKS automatically captures measurement data about the work as it progresses, thanks to the ProjectCatalyst reporting discussed above, and it provides relational calculus query capability to compute statistics. PEAKS also gathers pre-specified quality metrics (such as code complexity), which can be interpreted in the context of one or more quality frameworks adopted by the project (such as the RADC Quality Framework). Amadeus provides a repository for capturing arbitrary metrics data, together with a set of analysis and reporting aids.

4.2 History and Current Status of the PSE

The STARS components of the Demonstration Project PSE (PEAKS and ProjectCatalyst) were introduced in late 1993 and early 1994. A major pilot was conducted in March of 1994, and based on the results of the pilot, the project began use of these tools for the SCAI Release 2 Specification activity. Both the pilot and the operational use of these tools provided substantial practical experience on both the specific implementation of the tools and the process-driven paradigm supported by them. The main problems that emerged were:

- The toolset was not sufficiently stable to support production work;
- PEAKS and ProjectCatalyst were not fully integrated, resulting in redundant manual work to translate PEAKS models to ProjectCatalyst process performance pages;

[13] *PEAKS is the commercial name chosen by ccPE for the STARS-supported process modeling facility formerly known as Software Process Management System (SPMS). Since the name change is quite recent, most STARS publications, including [DemExp95] use the SPMS name.*

- ProjectCatalyst tool and database administration was excessively complicated; and
- The ProjectCatalyst implementation of the "State Data Repository" (SDR), originally designed to support text files, was not sufficient for the project's heterogeneous artifact types or its configuration management requirements.

These problems were largely due to the unprecedented nature of the usage paradigm being supported by the tools, and a resulting under estimation in the complexity of the required software.

Despite the problems, the project decided that there was sufficient promise in the toolset to plan a major upgrade. The upgrade, targeting all but the last of the above problems, was delivered on schedule in November, 1994, and is now being used for the SCAI Release 3 Application Engineering Specification activity. The fourth problem - the need to provide a more mature artifact management capability - remains a future objective.

During this same period, an affiliated team, funded by the Air Force's Sacramento/ALC EISE program at Sacramento, California, was conducting a feasibility study on the development of a tool to support both $IDEF_0$ activity-based process modeling and $IDEF_{1X}$ data modeling. This work yielded the ProDAT tool, based on the VSF[14] engine. After conducting a pilot of ProDAT in September, 1994, the project decided to use ProDAT in place of DesignIDEF and Framemaker, to reduce the complexity of assembling IDEF-based process descriptions, from a tool with a single database, as opposed to the more manual intensive approach that was being employed.

Figure 9, on page 21, depicts the PSE in use on the Demonstration Project as of this writing.

4.3 LESSONS LEARNED

The lessons presented in this section are derived from roughly five years of experience in studying process and process automation technology, and piloting a succession of Process Support Environments (PSEs).

- Building a Process Support Environment is an ambitious undertaking.

  We underestimated the difficulty in building and transitioning the STARS portion of the Demonstration Project PSE (PEAKS and ProjectCatalyst/Process Weaver).

  In retrospect, there are several reasons for the underestimation:

[14] ***Virtual Software Factory (VSF), provided by VSF, Inc., is a tool specifically designed for developing graphically-oriented computer-aided software engineering (CASE) tools.***

- Despite prior experience with PSE antecedents (refer to the Appendix to this paper), the usage paradigms and ingredient technologies were still largely unprecedented.

- Prior experience (e.g., with the Cleanroom Engineering Process Assistant[15]) was in simpler usage environments and with a smaller user population. The Demonstration Project SEE requirements were more complex than envisioned, due to such factors as the heterogeneous hardware and software environment and use of multiple SEE frameworks.

- Work prior to the identification of Loral's Demonstration Project partner was geared to the Cleanroom software engineering process - which had already been specified, tooled, and piloted with predecessor PSEs. Although the new organization decided to incorporate Cleanroom principles in their process, it turned out that most of the prior work had to be rethought.

  This factor relates to a larger project-level lesson learned cited in [DemExp95]: a project must work out its own product-line process - organizations will seldom be able to use an off-the-shelf process defined by another party.

- Attempting to make a large number of significant approach changes on a project can impede the progress made in any one area.

  This is another example, cited in [DemExp95], where larger project-level issues affected work in the SEE area. The Demonstration Project as a whole had very ambitious technology objectives, resulting in simultaneous development of basic approaches on several fronts at once, including:

  - Process definition methodology,
  - Process content (numerous advanced processes are targeted by the SWSC, including domain engineering, product-line configuration management, metrics, etc.), and
  - Process automation.

  Adding to these approach issues were the engineering issues associated with building the SCAI application itself. Although the project has done remarkably well at synthesizing a promising approach in nearly all of these areas, each one of them suffered to some extent from unknowns and variables - as well as from competition for intellectual energy.

- Transitioning to a significantly new approach requires an incremental strategy.

[15] *Cleanroom Engineering Process Assistant; described in the Appendix on page 34.*

Another lesson we re-learned was that of technology transfer techniques. When it comes to radical new ideas, such as process-driven development, baby step introduction of new technology, followed by successful technology use is far better than trying to introduce too much technology, too soon. Once this approach was taken, technology introduction and adoption was greatly facilitated. Performing pilot products to permit a customer to gain experience with a new technology is vital to the adoption process.

- There are additional SEE integration challenges to be addressed to improve the practicality of a PSE.

  While we expect the PSE concept to pay rich rewards in the long-haul, there are some integration challenges ahead. We believe the following process integration lessons - and the experience upon which they are based - represent one of the most significant results of the investment in process technology made by both STARS and the Demonstration Project organization.

  – Differences between platforms, sometimes quite subtle, can complicate integration.

    For example, one of our integration problems was the control and data integration of PEAKS and CAT/Compass[16]. A message set was carefully planned to support data exchange between PEAKS and CAT/Compass. The testing of the individual message sets against the two tools was also successful. However, the control integration mechanisms tailored on one environment (SUN UNIX) exhibited different behavior on the other (IBM AIX). Problems in permitting full message exchange between these two tools were never fully resolved.

  – Integrating a new tool as a means of gaining "off-the-shelf" technology is often more complicated than anticipated.

    As an example, we sought to integrate a commercial project management package with PEAKS so that we could realize "process-driven planning" without implementing planning functionality in PEAKS. We soon found that we had to define a fairly complex interface specification, because the tools' perceptions of apparently similar data turned out to be quite dissimilar when examined in detail.

  – The "multiple database problem" is a pervasive integration issue.

    First, if two related tools are left un-integrated, the user is forced to manually keep the respective databases in agreement. Since this

[16] *CAT/Compass is a management planning package from Robbins-Gioia. CAT and CAT/Compass are registered trademarks of Robbins-Gioia, Inc. of Alexandria, Virginia*

can be a formidable challenge, one or the other database is likely to fall behind, diminishing or nullifying the value of the associated tool. Second, when integrating two tools with overlapping functionality, each side must understand the other's database so that they can formulate a strategy for keeping each other's data in sync. Third, given that integrating the two tools is feasible, careful design is needed to be sure no messages are lost.

- Integrating separately developed process support tools can involve modeling paradigm issues as well as SEE integration issues.

  In view of the prior lesson, the Demonstration Project would ultimately like to smoothly integrate all of the PSE capabilities depicted in Figure 9 on page 21. One of the key issues in achieving this integration is the differing modeling points of view taken by the tools.

  The Air Force is mandated to use the $IDEF_0$ notation for activity modeling, and the SCAI project opted at the outset to use MetaSoft's DesignIDEF tool for developing its process framework model. Unfortunately, despite its strength for understanding activity relationships, $IDEF_0$ is not sufficient for specifying enactable processes, because it is not capable of specifying constrained activity flow and control.

  PEAKS, part of the STARS toolset, does model constrained activity flow and control, but does not have an $IDEF_0$ view. Thus, the objective of integrating the two databases was thwarted not only by tool issues, but more importantly, modeling paradigm issues.

  Although this remains a long-range issue, two improvements are already underway. The first is the conceptual integration of the $IDEF_0$ view into a consistent enactable process specification in the STARS/SEI Process Definition Information Organizer Templates, worked out in cooperation with the Demonstration Project [Ett3-94].

  The second is the transition to a new prototype IDEF-based modeling tool called ProDAT. ProDAT first integrates $IDEF_0$ activity modeling with $IDEF_{1X}$ information modeling. It then adds a new artifact state transition modeling view and uses the new view to generate process sequencing.

  Assuming the new ProDAT paradigm proves productive for the Demonstration Project, it may be possible to pursue tool integration between ProDAT and PEAKS.

- Message-based control integration services require careful design to assure key data is not lost.

For control integration among PSE components, we chose to use broadcast message server capabilities provided in IBM AIX SDE Workbench. The BMS metaphor is that a sender places a message on a software bus and one or more listeners hear the message and take appropriate action.

A central precept of process integration is that information used to support process control decisions must not be lost. Implementing PSE integration using BMS proved to require more engineering than expected, because message delivery is not assured. Use of BMS for process control requires considerable attention to such protocol issues as assured start up of listeners, handshaking, and error recovery. Redesign of our integration has led to a much more robust interface, but the potential still exists for message loss.

There were other limitations in Workbench that surfaced during the SEE integration effort. For example, communication between tools was supported only within a single user session. We took advantage of Process Weaver's inter-user messaging capability to help offset this problem.

The IBM Workbench product is an implementation of the Hewlett-Packard Broadcast Message Server (BMS) technology. We believe many of the Workbench limitations were the result of the lagging implementation of BMS improvements.

The emerging standard for SEE integration messaging is found in Sun's ToolTalk. This technology is part of the emerging industry standard referred to as the Common Desktop Environment (CDE). All the major environment platform vendors (Sun, Hewlett-Packard, IBM, Digital, and others) have agreed to provide CDE compliant products with their platforms. Future releases of AIX should provide an implementation of BMS (through ToolTalk) that will ease most of the limitations that we encountered.

- Data integration is impeded by toolsets that provide monolithic artifact management functions.

  In today's market, most toolsets that provide large amounts of functionality often perform self-contained artifact management - from their own points of view. In part, this stems from the absence of agreed-upon standards for common artifact management. The resulting mismatches make it difficult to implement a coherent artifact management encapsulation layer (as shown in the conceptual model of the SEE in Figure 3, on page 9). On the Demonstration Project, ProjectCatalyst was delivered with a self-contained "State Data Repository" (SDR), which both stored artifacts and maintained artifact states from a process point of

view. Rational Apex, used for Ada code development, manages its own artifacts, greatly enhancing its performance due to its detailed understanding of Ada semantics. Apex also implemented a much more sophisticated Configuration Management strategy than the SDR. Finally, the project selected IBM CMVC for its general-purpose CM solution, since it provided problem tracking capability, which Apex lacked. The net is that there are three artifact management methods available on the SEE. Currently, the project has decided to avoid use of the SDR entirely, and it is working on a locally written integration between CMVC and Apex.

Ultimately, as discussed out in [Randall89], what is needed is a general-purpose artifact management repository for the SEE - together with a standardized API - so that projects could replace one repository implementation with another as better solutions appeared. This is an active area of R&D, and no agreed upon approach has emerged. In the meantime there seem to be only two solutions for toolset vendors: continue to pursue independent monolithic implementations, or define a minimum essential API for an abstract repository and attempt to adapt one or more commercially available repositories to that API.

- Process state management should ultimately be handled by a central SEE server.

  Our experience has shown us that ideally, process state should be managed by a server that all process support tools may access, and selected tools may update. The concept of the process server was also independently arrived at by the Arcadia project [Heimbigner95]. Failing an effective process server, a single tool should ideally manage and provide process state data to all requesting tools.

- Generic instantiation techniques can greatly reduce the need for process programming.

  The PSE evolution work leading to the LORAL STARS PSE toolset (described in appendix A) brought forth the idea of an adaptable process support environment layer with a defined set of APIs that could be easily interfaced to employ SEE integration mechanisms. In terms of the PSE model shown in Figure 7, on page 16, these innovations apply to the interface between the Tactical Planning/Coordination (TPC) and the Task Execution (TE) capability sets. In the case of the STARS toolset, TPC capabilities are provided by ProjectCatalyst and TE capabilities are provided by Process Weaver.

  Our approach to instantiating this interface required:

- The identification of a small set of generic process programs that support individual or collaborative activities that can be instantiated from planned tasks. One of the clear successes from our work is the recognition that all processes have common characteristics which permits the development and instantiation of generic process programs. These generic process programs were derived from a generic architecture for process enactment.[ETT2-94]. We have effectively used this concept to dramatically reduce our need to develop customized process programs. In fact, in our use of a generic architecture to support process enactment, we have employed one of the key concepts of megaprogramming by 16specifying a product line of process programs for use by the Task Execution Support capability of the PSE layer.

- The development of systematic techniques for binding SEE tools and services to a process program at task execution time, rather than in an off-line build procedure.

- A Process Support Environment is a powerful SEE integration vehicle.

  The main thesis of this paper is that a PSE is one of the most useful SEE integration vehicles, since two forms of integration are simultaneously at work: integration of the organization's process with the SEE, and integration of tools and artifacts within the context of the process.

5. CONCLUSION

We contend that it is becoming increasingly important for organizations - particularly megaprogramming organizations - to think of process and SEE as intimately tied. With the collaboration with the Air Force Demonstration Project team, we have gained valuable experience in integrating process and SEE and have identified a potentially profound SEE abstraction layer - the Process Support Environment - which has been the focus of this paper.

Section 3, Process-Based SEE Integration, presented the motivation for the PSE and provided a model for its structure and interfaces. The model identifies five cooperating "capability sets" within a PSE:

- Process Modeling;
- Project Planning, Monitoring and Control;
- Tactical Planning/Coordination;
- Task Execution; and
- Process Improvement Analysis.

Our work has provided experience in implementing and integrating each of these capability sets.

We also believe that our experience has yielded many pragmatic lessons learned that should prove useful to others pursuing process and SEE integration. Many of these lessons are cited in Section 4.3 of this paper.

Our priorities for future work are:

- Refine our model for the PSE layer (see Section 3.3, on page 14) as well as the approaches for applying SEE framework services to implement the layer.
- Develop minimum essential APIs for general-purpose artifact management and process state management repositories, and assess the viability of adapting current repository methods to serve these APIs.
- Continue to collaborate with both process and SEE researchers and practitioners, to help accelerate the convergence of the two disciplines.
- Continue to work with the Air Force to help them achieve the Demonstration Project's objectives and to capture experience. Specifically:
  - Actively participate in the "Process Engineering Support Team", designed to support the process users in following and improving the defined process - as well as to evolve the project's use of PSE automation capabilities;
  - Assist with the definition, instrumentation, and analysis of measurements to support process improvement; and
  - Make further improvements to existing PSE capabilities as funding permits.
- Continue to review process automation progress by other projects and commercial vendors.
- Participate in standardization efforts currently being pursued by industry groups, such as the Work Flow Management Coalition, to establish a reference model and API description for process work flow engines - described in this paper as the Task Execution Capability Set (see page 19).

REFERENCES

[Bristow95] Bristow D.J., Bulat R.G., Burton R. Product Line Process Development, Proceedings Seventh Annual Software Technology Conference, Salt Lake City, UT, April 1995. (Published simultaneously with the present paper)

[Bulat95] Bulat R.G. Space and Warning Systems Center Domain Engineering Experiences, Proceedings Seventh Annual Software Technology Conference, Salt Lake City, UT, April 1995. (Published simultaneously with the present paper)

[DemExp95] Air Force/STARS Demonstration Project Experience Report, Version 2.0 (Draft), CDRL Sequence A011-002D, Electronic Systems Center, AFMC, USAF, December 1994 (currently under review by the Government).

[Ett-92] Ett, W.H., R.H. Cobb, A. Kouchakdjian, Cleanroom Software Process Case Study: Lessons Learned from STARS Task IS-15, IBM FSC Technical Report 85.0165, IBM Federal Systems Company, Gaithersburg, MD, June 26, 1992.

[Ett1-94] Ett, W.H., S. Becker, Evaluating the Effectiveness of Process Weaver as a Process Management Tool: A Case Study, Proceedings of the Third Symposium of Assessment of Quality Software Development Tools, Washington, D.C, June 7-9, 1994.

[Ett2-94] Ett, W.H, R.H. Cobb, A. Kouchakdjian, Lowering the Entry Barrier to Process Execution Support, Loral Federal Systems Internal Report, Gaithersburg, MD, October, 20, 1994.

[Ett3-94] Ett, W.H, Phillips, R.W. SCAI Process Definition Training Package, Feb 1994, (available from the authors).

[Heimbigner95] Heimbigner, D., ProcessWall Process State Server, Demonstration Abstracts, ARPA Software Environments Technology Conference, Arlington, Virginia, January 5-6, 1995.

[NGCR93] Reference Model for Project Support Environments, Version 2.0 (Draft), Next Generation Computer Resources, 2 September 1993.

[NIST93] NIST Special Publication 500-211, Reference Model for Frameworks of Software Engineering Environments (Technical Report ECMA TR/55, 3rd ed.), National Institute of Standards and Technology, August 1993.

[Randall89] Randall R.L. xSPER - An Approach for Generating Extensible Integrated Project Support Environments, PhD Dissertation, University of California, San Diego, April 1989.

[Randall95] Randall R.L., Ekman R.G., Kent S.P. (Capt. USAF), Turner G.S. Integrating a SEE for Megaprogramming: Lessons Learned, Proceedings Seventh Annual Software Technology Conference, Salt Lake City, UT, April 1995. (Published simultaneously with the present paper)

[STSC94] Hanrahan R., Daud C., Meiser, K., Peterson J. Software Engineering Environment Technology Report, Software Technology Support Center, OO-ALC/TISE, Hill AFB, Utah, April 1994.

[SchMel92] Shlaer S., Mellor S.J. Object Lifecycles, Modeling the World in States, Yourdon Press, 1992.

[Sutton94] Sutton, S., personal communication, November 29, 1994.

[Trimble94] Trimble J. (ed.), STARS Program History 1983-1993 Version 1.1, available from the STARS Program Office.

[Young91] Young, P.S., R.S. Taylor, Team-Oriented Process Programming, UCI Technical Report 91-68, Department of Information and Computer Science, University of California, Irvine, CA, August 28, 1991.

AUTHOR BIOGRAPHIES

Dr. Richard L. Randall

Dr. Randall is currently working on the ARPA STARS program as the on-site lead for the Air Force/STARS Demonstration Project at Peterson AFB, Colorado. He provides consultation to the Air Force on megaprogramming technology transition issues and on long-range strategies for enabling a future product-line for the Space and Warning Systems Center. His area of technical focus on the Demo Project is the integration of the Demo Project Software Engineering Environment - notably the aspects dealing with the organization's process and artifact management.

Dr. Randall's research interests include software engineering methods and integrated project support environments (IPSEs). He has over 25 years of experience in all aspects of software engineering for large real-time systems on projects such as Gemini, Apollo, Safeguard, Sea Nymph, and B2. During this time, he has focused on the integration of methods and tools into the software process, and he has played lead role in various division-level software and systems engineering steering groups.

Dr. Randall received a BS in Mathematics from MIT and a PhD in Computer Science from UCSD. He is a member of the IEEE Computer Society and the ACM.

Mr. William H. Ett

Mr. Ett is currently working on the ARPA STARS Program, where he specializes in the development and transition of techniques for defining enactable processes, and the specification, design and development of automated process support applications. Mr. Ett's accomplishments include the design of the Cleanroom Engineering Process Assistant, the co-invention of the ProjectCatalyst front end and its generic process programming paradigm, the design of the initial LORAL STARS process support environment, and the co-development of the "STARS/SEI Process Definition Information Organizer Templates." Mr. Ett has over twenty years of experience in the development and delivery of government and military information processing and real-time systems.

Mr. Ett's research interests include the design and development of process support technology and tools to assist organizations in benefiting from process-driven software development, as well as the application of artificial intelligence techniques to support systems and software engineering.

Mr. Ett received a BS in Computer Science from the University of Maryland. He has also done graduate work in Computer Science and Operations Research.

APPENDIX: HISTORY OF PSE EVOLUTION

As a supplement to the main paper, this appendix provides a brief examination of the evolutionary path to the current Process Support Environment (PSE) and our conceptual model of the Process Support Environment overlay SEE layer. This will aid in understanding the description of the Demonstration Project PSE described in Section 4 of the paper.

We begin by discussing three predecessor STARS implementations, and we conclude by describing a representative contrasting approach being pursued by the Arcadia Project.

PREDECESSOR STARS IMPLEMENTATIONS

Starting in 1990, STARS evaluated tools and prototyped several environments and applications to support the process-driven development of software. This evolution was carried out in three phases:

- Development of a prototype Process Support System (PSS) to guide a project team in following a defined process. This phase led to the implementation of the Cleanroom Engineering Process Assistant (CEPA) [ETT-92]
- Development of a prototype PSS, using an interpretative process design tool to implement process support applications. This phase led to the implementation of the Process Weaver CEPA prototype [ETT1-94]
- Design and development of a pilot Process Support Environment (PSE), integrating the process definition, design, enactment and measurement capabilities developed on STARS [ETT2-94]. This pilot was actually used on the Air Force/STARS Demonstration Project. The resulting experience identified several required improvements, leading to the current Demonstration Project PSE.

The first two phases provided the user organizations with Process Support Systems - i.e., end-user systems that guided and supported practitioners in executing the organization's process. Process engineering activities (such as process definition and process programming) were performed by the PSS developers.

The third phase, yielding the current Air Force/STARS Demonstration Project toolset, provided the user organization with a full Process Support Environment (PSE) -

which allowed the organization both to perform its own process engineering activities and to produce and evolve its own PSS as well.

We now believe the ability for an organization to perform a full range of process engineering activities is essential to megaprogramming.

CLEANROOM ENGINEERING PROCESS ASSISTANT (90-94)

Implementing the CEPA prototype provided the Loral STARS team with its first experience in process integration. CEPA was implemented using the KI Shell[17]. KI Shell provided a method editor to support the design of a process support system, and provided extensive library services to support the implementation of the final system using the C programming language. The resulting CEPA process support system was not easily modifiable, as the work flow for the process was embedded in the application. Thus, system modification and re-compilation was required to make a change to the system. Where processes are well understood and do not have a great deal of volatility, implementing the process support system as a compiler has performance advantages.

To accommodate changes in the SEE, CEPA routines were designed to invoke UNIX shell scripts to perform designated functions. These shell scripts could be modified without affecting the system. KI Shell supported integration with presentation and data integration mechanisms. Control integration mechanisms were not available at the time of CEPA's development. Thus CEPA was not only a process support system, but contained several of the capabilities allocated to the Process Support Environment layer.

Implementation of CEPA required a complete life cycle of software development activities from specification through implementation and testing. There are no short cuts.

The initial CEPA prototype was refined and fielded for use at the US Army's Picatinny Arsenal from 1992 through 1994, where it supported the MBC upgrade project. CEPA's field test experience demonstrated that process support systems could be built to guide a team of development engineers through a defined process, while performing their work.

Although CEPA did meet user's expectations, several issues were identified from its use:

- The system as implemented had limited scope, and extending the system meant additional software design, implementation and testing.
- CEPA had poor facilities for technical task planning and dispatching. These functions were supported using a project management system, followed by the use of a spreadsheet program.

---

[17] *KI Shell is a registered trademark of UES, Inc. of Dayton, Ohio*

- CEPA tasks, once dispatched were impossible to kill, thus the entire work step sequence for a task had to be walked through its completion, to delete it.

Table 1 provides a summary of the Process Support Environment Capabilities included in CEPA.

PSE Capabilities	PSE Tools	PSE Integration
• Process Modeling	• None	• None
• Project Planning, Monitoring and Control	• None	• None
• Process Performance Management	• Limited	• Yes, an editor was provided to add black, state and clear boxes as required. However, planning tasks required support external to the tool.
• Task Execution Support	• KI Shell Enactment Engine	• Yes, the enactment engine managed all work flow activities for the program.
• Process Improvement Analysis	• KI Shell Instrumentation Facilities	• Yes, task steps could be instrumented to collect and post measurement data.
• Process/Artifact State Management	• KI Shell State Object Manager	• Yes, KI Shell used a frame information management system to maintain state and attributes on all task and data objects.
• Collaborative Development Support	• KI Shell Messaging Services	• KI Shell permitted the implementation of process programs which supported collaborative development.
• PSS Development	• KI Shell Method Editor • C Language Compiler • Oracle	• None.

Table 1: CEPA Process Support Environment Dimensions

PROCESS WEAVER CEPA PILOT (92-93)

Implementing selected key functions of CEPA as Process Weaver cooperative procedures (or process programs), gave the Loral STARS team experience using an interpretive work flow management capability. The Process Weaver CEPA prototype was implemented using Process Weaver's process program development editors, namely, the cooperative procedure editor and the work context editor. The cooperative procedure editor permitted a process engineer to define work steps to perform a task as a petri net. Actions to support petri net processing were implemented using a script

language named CoShell, which provided a LISP-like capability for manipulating objects in a CoShell program.

Process Weaver's process integration mechanisms permitted the use of control integration mechanisms, namely HP SoftBench's Broadcast Message Service (BMS) to support application invocation and transaction message passing, and Weaver BMS, a Process Weaver implementation of BMS written to support the development of process programs which support collaborative development. Presentation integration mechanisms for process programs were managed through the Process Weaver facilities, while HP SoftBench's presentation integration mechanisms were provided for SEE engineers to encapsulate applications to be directly invoked from the SEE or from task work steps.

Process Weaver did enable the suspension or deletion of work tasks without requiring the work task to be completed.

Although Process Weaver solved a few of the concerns identified during CEPA use, it did present new issues:

- Process programs implemented using the Process Weaver cooperative procedure editor were easy to develop, and easy to change. However, change management of these process programs became problematic. Further, unless the process programs were properly managed, process practitioners had the ability to modify the process programs.
- The Process Weaver facilities available for supporting task planning, dispatching and monitoring were not effective and had to be managed outside of the Process Weaver environment.
- As in CEPA, process programming of customized applications was required. The impact of this is that customized software has to be specified, designed and implemented to support the development of software.
- Process State and History is only maintained for local executing task threads, as opposed to CEPA which provided access to global task state information to support inter- and intra-task process control.

Table 2 provides a summary of the Process Support Environment Capabilities included in the Process Weaver CEPA prototype.

PSE Capabilities	PSE Tools	PSE Integration
• Process Modeling	• None	• None
• Project Planning, Monitoring and Control	• Microsoft Project	• None. Microsoft Project plans could be exported as a flat file for use by Process Weaver.
• Process Performance Management	• Limited	• None. Task dispatching could be implemented to support Process Weaver's capabilities. However, planning and tracking these tasks required support external to the tool.
• Task Execution Support	• Process Weaver Enactment Engine using the CoShell Interpreter	• Yes, the enactment engine managed all work flow activities for the program.
• Process Improvement Analysis	• KI Shell Instrumentation Facilities	• Yes, services were provided to instrument task steps to collect and post measurement data, or to invoke external programs to collect data.
• Process/Artifact State Management	• Process Weaver CoShell Facilities	• Yes, Process Weaver maintains local state data for each process program (cooperative procedure) launched. Artifact state information must be programmed into the process program for it to be maintained.
• Collaborative Development Support	• Process Weaver BMS	• Process weaver permitted the implementation of process programs which supported collaborative development.
• PSS Development	• Process Weaver Cooperative Procedure and Work Context Editors (Process Design) • CoShell language and interpreter	• Yes, CoShell is the mechanism for executing Process Weaver process programs.

Table 2: Process Weaver CEPA Prototype Process Support Environment Dimensions.

PILOT LORAL STARS PROCESS SUPPORT ENVIRONMENT (93-94)

The pilot process support environment was implemented to provide an environment to support the activities of the process activity life cycle, namely 1) the specification of software processes, 2) the design and implementation of process

programs, 3) the execution of process programs, 4) the collection of measurement data and 5) the analysis of that data to support process improvement. These activities are supported by the PSE capabilities introduced in section 3.3

To support Process Modeling and Project Planning, Monitoring and Control, the STARS-developed Process Engineering and Kernel System (PEAKS)[18] was selected. PEAKS was programmed to respond to control messages sent from either HP SoftBench's BMS or through PEAKS's API. Process state and history information was maintained by Oracle.

To support Process Performance Management, a new set of capabilities had to be developed. To satisfy the requirements for this capability set, ProjectCatalyst was implemented. ProjectCatalyst provided a capability to support the delegation of processes and the planning, dispatching and tracking of tasks required to support the requirements of a delegated process. ProjectCatalyst permitted technical managers and task leaders to plan and organize tasks using a spreadsheet paradigm, and to use this same spreadsheet view to assign, dispatch and track work tasks.

ProjectCatalyst was programmed to both send and receive control messages from PEAKS via HP SoftBench to enable ProjectCatalyst to query PEAKS about planned processes and tasks, and to post actual performance data on process and task status to PEAKS. ProjectCatalyst maintained process and task state and history data in Oracle. ProjectCatalyst was also programmed to launch process programs for execution by Process Weaver, where ProjectCatalyst controlled the work to be managed by Process Weaver. ProjectCatalyst made use of Process Weaver's process integration mechanisms for launching work tasks and passing control messages between Process Weaver users.

To support Task Execution Support, Process Weaver was selected. As stated above, Process Weaver was interfaced with ProjectCatalyst, where ProjectCatalyst controlled the invocation of Process Weaver process programs and Process Weaver controlled their execution. Process programs could be instrumented to invoke SEE tools and services, collect measurements or call measurement collection agents, and to manage artifacts.

No capabilities were selected to support Process Improvement Analysis for the Pilot PSE, other than measurement data collection.

The Pilot Loral STARS PSE included several unique features:

- It made use of a generic process programming paradigm for implementing process programs. This reduced the need for customized process programming and provided greater task management flexibility for Process Performance Management.
- It provided a flexible mechanism for planning, tracking, dispatching, and canceling tasks.

[18] *Formerly known as Software Process Management System (SPMS); PEAKS is ccPE's commercial name for the product.*

- It provided consistent process state and history data among applications - made possible through data integration using an Oracle database.

From this first integration attempt, and from the essential usage experience that resulted from trial usage on the Air Force/STARS Demonstration Project, several issues were identified that needed to be addressed. Based on joint discussions of priorities, some of these issues (identified in the following list) were targeted for a major upgrade - leading to the current Demonstration Project PSE:

- Integration between PEAKS and ProjectCatalyst needed to be improved to reduce the amount of manual work required to keep the two environments in agreement. Although there were provisions for automatic reporting from ProjectCatalyst back to PEAKS, there were no provisions for automatically initializing ProjectCatalyst process information from the PEAKS database.

  (The integration is substantially improved in the current version. PEAKS exports plan data and ProjectCatalyst initializes its database accordingly.)

- Architecture services need to be developed to interface with heterogeneous SEEs to access their data and employ their services. Standard programs for providing wrappers around said services are required to facilitate application program integration into process program workflow activities. Basic problems also lie in data/file/program object ownership and poorly understood access protocols.

- Data predetermination for tasks is not sensible. The task planner is forced to anticipate data requirements to support the task prior to it being dispatched. Data objects must be assignable, either prior to task dispatching or during task execution. Further, the task performer must have access to all relevant data objects to support task work.

- Execution requirements for PSE components were too complex for the average user. User front-end programs are required to be developed to perform all housekeeping chores for PSE users.

  (A "front-end" script has now been provided to handle this automatically.)

- PSE administration is enormously complex. PSE administration programs are required to ease the burden on site personnel for both PSE installation and PSE administration.

  (A set of administrative scripts is now available that greatly simplifies administration.)

Table 3 provides a summary of the Pilot Loral STARS Process Environment Capabilities.

PSE Capabilities	PSE Tools	PSE Integration
• Process Modeling	• PEAKS	• Partially. Manual binding of plan process ids with ProjectCatalyst process pages. Automatic data reporting once binding was established through BMS.
• Project Planning, Monitoring and Control	• PEAKS • CAT/Compass (Eliminated)	• Partially. CAT/COMPASS was integrated with PEAKS to support PEAKS Plan Scheduling and status data exchange through BMS. PEAKS later subsumed required CAT/COMPASS functionality.
• Process Performance Management	• ProjectCatalyst Prototype	• Partially. Mechanism for manually binding PEAKS process ids with ProjectCatalyst process pages. Automatic data reporting once binding was established through BMS. Mechanism for invoking Process Weaver process programs.
• Task Execution Support	• Process Weaver Enactment Engine using the CoShell Interpreter	• Yes, the enactment engine managed all work flow activities for the program.
• Process Improvement Analysis	• Process Weaver CoShell Facilities • Amadeus Measurement System (AMS) • PEAKS Measurement Quality Facility (MQF)	• Yes, services were provided to instrument task steps to collect and post measurement data, or to invoke external programs to collect data, such as the Amadeus Measurement System or the PEAKS MQF.
• Process/Artifact State Management	• Oracle • Process Weaver CoShell Facilities	• Yes, Oracle was used as a persistent state object store for both PEAKS and ProjectCatalyst. • Process Weaver maintains local state data for each process program (cooperative procedure) launched. Artifact state information must be programmed into the process program for it to be maintained.
• Collaborative Development Support	• Process Weaver BMS	• Process Weaver permitted the implementation of process programs which supported collaborative development.
• PSS Development	• Process Weaver Cooperative Procedure and Work Context Editors (Process Design) • CoShell language and interpreter	• Yes. The Process Weaver editors support code and go development of process programs.

Table 3: Pilot Loral STARS Process Support Environment Dimensions.

A CONTRASTING APPROACH

ARCADIA PROCESS SUPPORT ENVIRONMENT

The Arcadia Consortium has developed a unique set of capabilities to provide process integration capabilities for their SEE development efforts, as well as provide capabilities to develop process programs. We have chosen to discuss Arcadia's Process Support Environment capabilities, because of their natural fit within the capability sets we have defined for characterizing a Process Support Environment layer.

To support Process Modeling, a tool called Teamware [Young91] was developed. Teamware is also a process specification and programming system, but could be used to support the development of process models for implementation as APPL/A programs.

To support Project Planning, Monitoring and Control, a tool called ManLobbi was developed to support project planning. ManLobbi also supports the capabilities of Process Performance Management, by supporting detailed task planning and task dispatching.[Sutton94].

To support Task Execution Support, process programs must be implemented in the APPL/A, a capability for precompiling APPL/A process programs into Ada. The resulting compiled Ada programs support process program execution.

As uniform maintenance of process state and history is important to support and coordinate the execution of many different process programs, Arcadia developed a capability called the Processwall Process State Server [Heimbigner95] to provide for the storage of process states, as well as for operations for defining and manipulating the structures of those states.

This discussion does not include all of the components of the Arcadia SEE, but addresses the relevant key components. It should be noted that these components are the result of University research and as such are not commercially available.

Table 4 provides a summary of the Process Support Environment Capabilities of the Arcadia SEE.

PSE Capabilities	PSE Tools	PSE Integration
• Process Modeling	• Teamware	• None with other Arcadia process tools.
• Project Planning, Monitoring and Control	• ManLobbi (developed for APPL/A use) • Teamware	• ManLobbi - Yes. Integrated to capture status from executing process programs.
• Process Performance Management	• ManLobbi	• Yes. Integrated to support the dispatching of tasks and the capture of status from executing process programs.
• Task Execution Support	• APPL/A Process Programs	• Yes. APPL/A process programs employ SEE presentation, data and control integration mechanisms.
• Process Improvement Analysis	• Amadeus Measurement System	• Yes. APPL/A programs can be instrumented to invoke external programs to collect and post data.
• Process/Artifact State Management	• Managed within APPL/A applications using either TRITON or ProcessWall.	• Yes. APPL/A programs can be programmed to employ the services of an object manager (TRITON) or ProcessWall to maintain process state data.
• Collaborative Development Support	• Process Weaver BMS	• Process weaver permitted the implementation of process programs which supported collaborative development.
• PSS Development	• Teamware (process design) • APPL/A (process program development)	• Teamware - Yes. Supports code and go process programming. • APPL/A - No.

Table 4: Arcadia Process Support Environment Dimensions.

Appendix I

SWSC Domain Engineering Experience

The material in this section is intended to supplement Section 2.0, Domain Engineering/Application Engineering (DE/AE). It consists of the complete text of a paper entitled "SWSC Domain Engineering Experience" [Bulat95], to be presented at the Software Technology Conference, STC '95.

The paper focuses on the Domain Engineering experience on the Demonstration Project.

Presenter: Brian Bulat

Title: SWSC Domain Engineering Experience

Track: Architecture (7)

Day: Tuesday

Keywords: domain engineering, architecture, reuse, object, class, framework, process, megaprogramming, product line.

SWSC DOMAIN ENGINEERING EXPERIENCE

This paper describes some of the lessons the Space and Warning Systems Center (SWSC) Domain Engineering team has learned while working on a Megaprogramming effort. In order to set the context for the lessons learned, the SWSC Environment, Domain Architecture, and Domain Architecture Framework are described. Architecture Framework extends the concept of architecture to include the relationship of architecture to other domain artifacts (system specifications, code, etc.) and the processes that support the construction of the architecture and artifacts. Finally, the lessons learned over the last two years on the Space Command and Control Architectural Infrastructure Project (SCAI) are presented. The lessons have been abstracted so that their scope is wider than just the SCAI project.

1.0 MEGAPROGRAMMING

The Software Technology for Adaptable and Reliable Systems (STARS) program under the Advanced Research Projects Agency (ARPA), has evolved the concept of Megaprogramming [Trimble94], or Process Driven, Product-Line Software Reuse supported by an integrated Software Engineering Environment. Megaprogramming identifies families of applications that share common traits and characteristics (a product line). Each common trait or abstraction may result in a common Ada component that can be used to construct multiple applications in the domain. Megaprogramming requires that software be produced according to defined processes. These processes are grouped into two lifecycles. A Domain Engineering Lifecycle, under the aegis of a Domain Engineering Organization, identifies common components; an Application Engineering Lifecycle uses the common components to construct specific systems. A Domain Management organization makes the cost decisions and scheduling decisions related to when common components are to be constructed and then integrated into individual applications.

2.0 SWSC ENVIRONMENT

The Space and Warning Systems Center (SWSC) in Colorado Springs maintains and modifies Command and Control Systems for NORAD, USSPACECOM, and AFSPC. Currently the SWSC is responsible for twenty-seven "stovepipe" systems, comprising over 10 million lines of code developed in twenty-four different languages on a variety of hardware platforms. This maintenance nightmare is a fertile environment for introducing Megaprogramming. At the SWSC, the SCAI (Space Command and Control Architectural Infrastructure) project has been instituting Megaprogramming for the past two years. The SCAI project is the STARS/Loral/Air Force Megaprogramming Demonstration Project out of ARPA.

The SCAI approach (see Figure 1) to the Product-Line Software Reuse aspect of megaprogramming, is to use a software process called Domain Analysis to create a Domain Specific Software Architecture, to which all individual systems in the domain/family are mapped. SCAI interprets the Domain and Application Lifecycles as being very tightly coupled: All Domain Analysis occurs in the Domain Engineering Lifecycle, but all component construction occurs in the Application Lifecycle. Because of this tight coupling, the Software Development Process is referred to as the Domain Engineering/Application Engineering Process (DE/AE).

SCAI intends to demonstrate that the SWSC version of Megaprogramming will increase software quality while decreasing the cost of developing and maintaining families of related SWSC Command and Control systems.

The SCAI project is creating a Space Tracking and Warning Application in the Space Domain using the SCAI Megaprogramming Technologies. Preliminary results in the first intermediate delivery of the SCAI System indicate that over 50% of the code in the system is reused or generated, just including the services layer of the architecture (explained in following paragraphs). It is presumed that as the scope of the domain is extended to encompass more than Space Systems, the percentage of generated and reused code will dramatically increase. Quality results have not yet been reported.

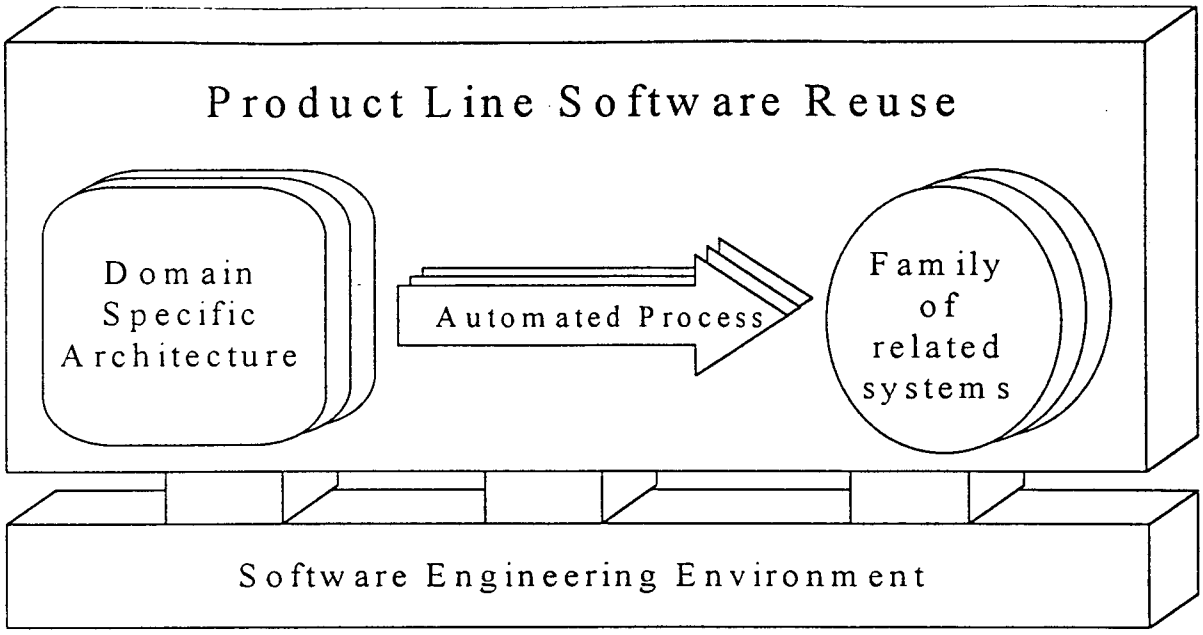


Figure 1. SCAI Megaprogramming

3.0 SCAI DOMAIN ARCHITECTURE

The SWSC attempted to determine a common architectural strategy for reengineering SWSC Command and Control Systems for several years prior to the SCAI project. This effort culminated in the RICC architectural infrastructure approach [Bristow93]. The so-called "Chip Diagram", Figure 2, represents the architectural concept at the start of the SCAI project. This concept had the SWSC C^2 system areas (Missile Warning, Space, Weather, Sensor Gateway, Air, and Intelligence) all being supported by a common infrastructure. The SCAI Project based the SCAI application on this infrastructure, which is enabled by the RICC tools.

The RICC, Reusable Integrated Command Center, is a set of run-time services supporting User Interfaces, Data Management, Message Handling, and Networks. Network services are a subset of system services in Figure 2. The RICC services are tailored to a particular system via a set of interactive Ada code generators, represented as Code Generator Tools in Figure 2.

This initial SCAI "chip" architecture was decomposed into a layered Domain Requirements Model (DRM) and a set of Application Architectural Models (AAM)s. The current scope of the DRM is the SWSC Space Domain; each AAM is specific to one system in the domain. The layered DRM is a modified Booch [Booch94] Class Model while the AAM is a network topology model and a mapping of application tasks to machines. These models, described below, comprise the ***SCAI Domain Architecture***.

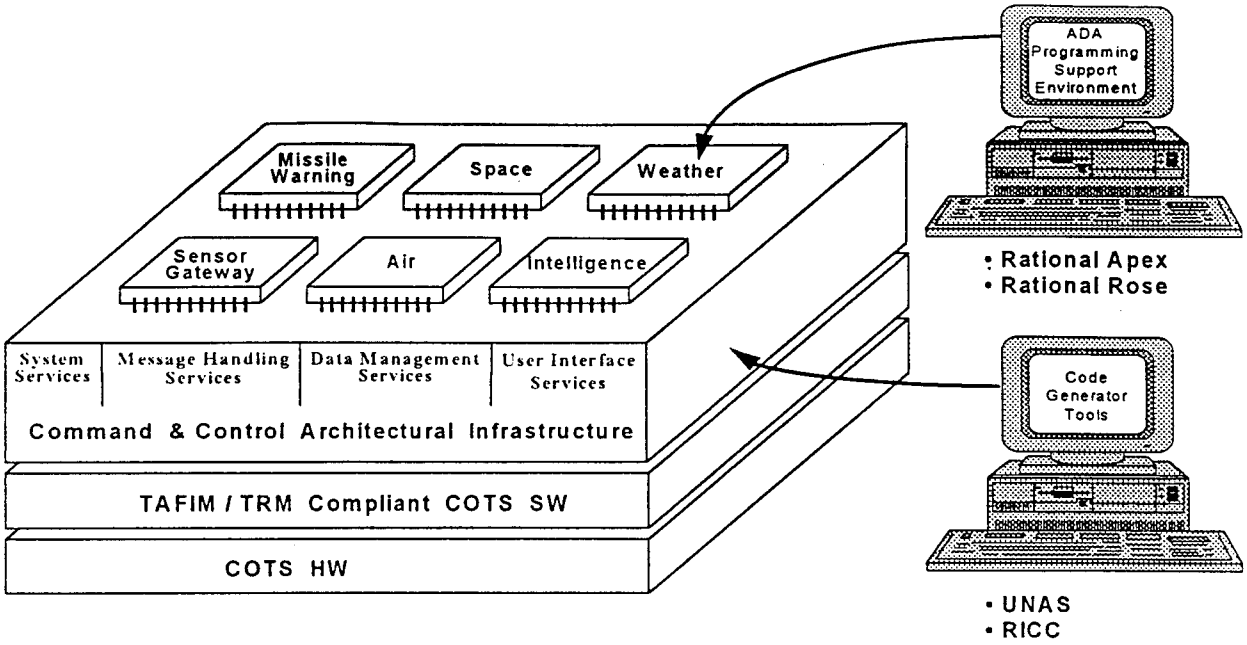


Figure 2. Initial C^2 Architectural Goal

3.1 DOMAIN REQUIREMENTS MODEL

The "chip model" was abstracted into a three layered object-oriented model called the Domain Requirements Model (DRM), Figure 3, by adding extra domain-specific model layers to identify types of reusable components other than the common service routines, and to further insulate the domain components from change.

3.1.1 Booch Model Composition

The DRM is a modified Booch [Booch94] Class Model. It contains a Class Diagram along with Class Templates and Object Scenario Diagrams. A Class Diagram identifies the static relationships between the classes in the domain, such as "a groundstation *tracks* a satellite", or a "maneuverable satellite is a subclass of satellite". Class Templates identify the details of class definitions: class operations (i.e., calculate a satellite orbit), class state data (i.e., satellite orbit definition), and class state transition diagrams (i.e., calculate a satellite orbit if new observations exist) if the class has complicated behavior. Object Scenario Diagrams depict single paths through the system, tracing the flow of control, as messages are passed from object to object, from one system boundary to the other. For instance:

(1) go to the groundstation object to get observations,

(2) pass the observations to the satellite object to calculate the new orbit,

(3) pass the new orbits to the orbit analyst object for viewing,

(4) put the newly calculated orbit in the satellite catalogue.

3.1.2 DRM Service Layer

The RICC Infrastructure Services, identified as the C^2 Service Layer in Figure 3, are services which are applicable to the whole C^2 domain. As new systems are analyzed and reengineered via the Domain Engineering / Application Engineering (DE/AE) process, the intention is to identify multiple instances of existing application services, and generalize them into a new C^2 Service Layer. These common Services act as servers to requesters (clients) in the layers above the C^2 Service Layer

COTS and GOTS products for use in the Service Layer are selected, as much as possible, so that they are compliant with the DoD TAFIM, Technical Architecture Framework for Information Management..

3.1.3 DRM Application Layer

Within the set of C^2 systems for which the SWSC is responsible, the Space Domain was chosen as the focus of the SCAI project. Common abstractions of the space domain are placed in the Application Layer. This Application Layer, shown as the middle layer of Figure 3, contains classes such as Groundstations, Satellites, and Observations, which are understandable to anybody familiar with the Space Domain. The classes imbed algorithms for things like orbit determination, and contain data such as orbits. Inheritance is used to identify system differences and commonalties in the domain. Commonalties are identified in "parent" classes while differences are identified in "child" classes.

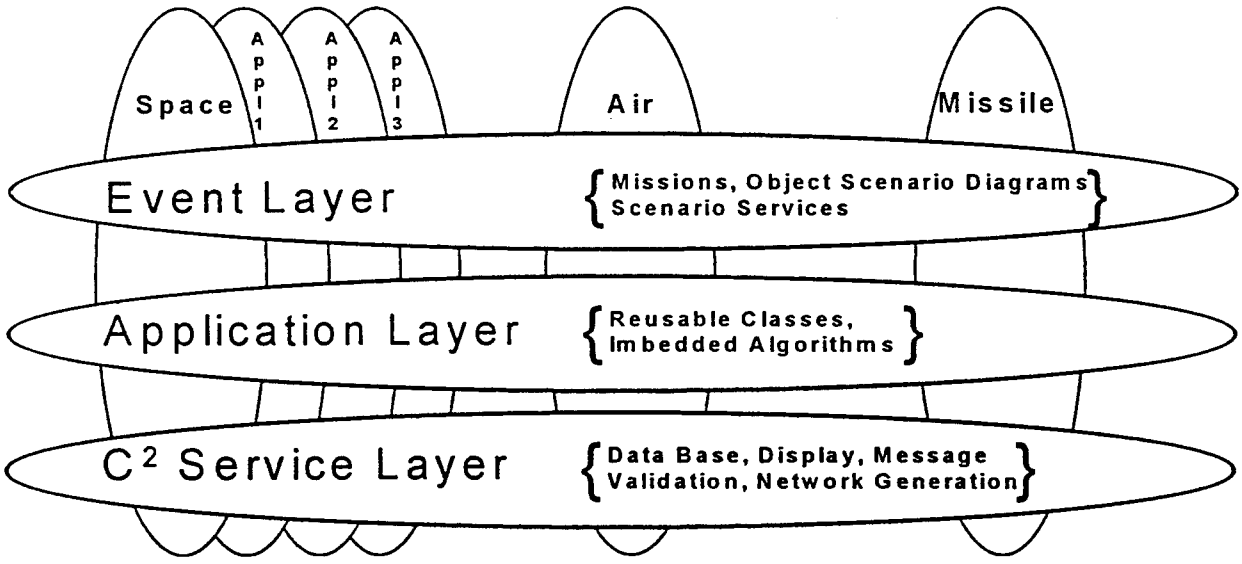


Figure 3. Domain Requirements Model

3.1.4 DRM Event Layer

Both the C^2 Service Layer and the Application Layer act as servers to the Event Layer. The Event Layer contains modified Booch Object Scenario Diagrams (OSD)s. Each class in the application layer may be included in many OSDs in the Event Layer. The OSDs act as a cross-check, to ensure that the classes defined in the application layer contain sufficient operations to handle all system functional requirements. A normal Booch OSD contains a single path through a system, tracing the flow of control, as messages are passed from object to object, from one system boundary to the other.

The SCAI DRM Event Layer is different than a set of Booch OSDs. Each Object Scenario Diagram (OSD), in the SCAI Event layer, traces a *set of paths* through the system, with messages connecting *all* the objects necessary to accomplish a single Space Mission. Space Mission examples are "Manually Determining an Orbit " or handling a "Satellite Reentry." These missions would typically be identified in a concept of operations document for a SWSC C^2 system, and would allow the Event Layer to be meaningful to Space Mission Experts. Each OSD, or mission, identifies every object needed by that mission. Each SCAI mission also relates to a meaningful, cohesive subset of the functional requirements for a system.

Each object within the OSD will eventually be transformed into an Ada Package or Generic Package, following SCAI coding rules. Therefore, each OSD also shows the necessary software needed to construct the Mission as an Ada Program, "on top" of the Service Layer run time services. Each SCAI OSD is a useful abstraction, understandable to a user of Space Systems, and shows how to construct an Ada Program to implement a SCAI Mission.

3.1.5 SCAI DRM versus other Domain Model Types

The DRM is a logical, machine-independent, representation of the common objects in all systems in the domain. So, DRM is really a misnomer. It is NOT a model of requirements, as one would never have included Services in a requirements model, because a requirements model is typically a model identifying a problem to solve, and not a design solution to that problem. Note also that the SCAI choice for a Domain Model is a high level design, including dynamic behavior as specified in the OSDs. Typical domain models [Diaz91-1] contain only static domain information. SCAI believes that a simple categorization scheme, embodied in a static model, does not provide enough information to determine if the reusable components (one component per class), can actually be used in the construction of the systems in the domain. The OSDs show how the classes (components) are used, and demonstrates that the components are sufficient to construct systems in the domain.

3.1.6 Reasons for DRM Layering

If the layering structure and rules for using the layers in the domain are consistent across that domain, then the DRM can be built iteratively. The characteristic

layering structure chosen for the DRM should be applicable to the whole SWSC domain.

The anticipated value of this layering is fourfold:

(1) The creation and isolation of a service layer allows the substitution of new, superior services in the future, while minimizing the impact to Space Application Layer Code. For instance, an object oriented data base could be substituted for the existing relational data base.

(2) The Service Layer makes it easier to be compliant with DoD TAFIM.

(3) New or modified missions can easily be created by reusing existing components in the Application Layer.

(4) The DRM layering scheme not only helps identify reuse opportunities, but also identifies the type of expertise that can be localized in product-line functional organizations, as explained in the following paragraphs.

Experts who understand the Missions in the SWSC domain need only understand the Mission Layer of the Domain Requirements Model, and need not be software engineers or system architects. These mission domain experts can collaborate to identify and abstract common missions across systems, and to project what new missions will be needed in the future.

Experts who understand orbits, trajectories, and the mathematics behind these topics, can be shared across appropriate SWSC supported missions, as part of a functional domain organization supporting the Application Layer.

Experts who understand typical software engineering tasks like relational data bases, or message parsing, can be considered as a pool available to all SWSC missions. They conceptually belong to the Service Layer of the SWSC, and should be responsible for upgrading and extending RICC services, or substituting services equivalent to those provided by RICC.

3.2 SCAI Relationship Between Domain and Application Models

SCAI literature refers to both Domain and Application Requirements Models (DRMs and ARMs). Currently there is only a single DRM, which is being used to construct the SCAI application. Referring to Figure 4, classes (and resultant Ada Packages) are either specific to a single application or general to more than one application. Differences between applications are captured using child classes. Similarities between systems are captured using parent classes. OSDs in the Event

Layer are identified as either common to the whole domain or specific to a single (or several) system(s). For example, the Manual Orbit Determination OSD is common to Systems A, B, and C. It uses an Observation object which is also common to all three systems. However, Manual Orbit Determination also uses a Sensor object, which has different children for Systems A and B.

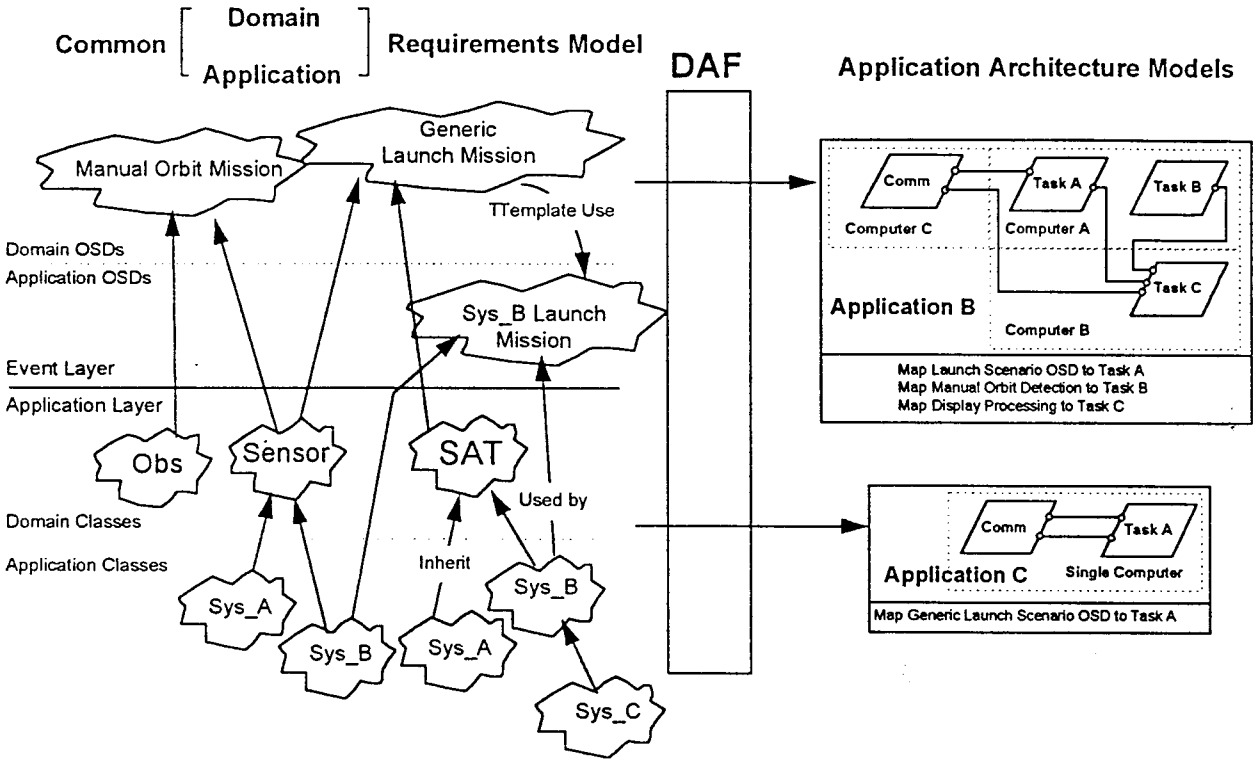


Figure 4. Domain/Application Model Relationship

The reason that application-specific classes are currently kept in the DRM is that when the next system is analyzed in the future, in order to extend the scope of the domain model, what is now an application specific class may be abstracted into a new class generic to several systems.

3.3 SCAI APPLICATION ARCHITECTURE MODEL

Referring to Figure 4, the Domain Requirements Model (DRM) is mapped into a set of Application Architecture Models (AAM)s, one per system in the Space Domain. Each AAM is comprised of:

- a UNAS network topology model
- a UNAS to Mission Mapping Table

A Universal Network Architecture System (UNAS) model [TRW94], also called a Software Architecture Skeleton (SAS), is a network topology, comprised of tasks and

circuits. Circuits identify the paths across which messages flow between tasks, or units of asynchronous work. The UNAS model also maps UNAS tasks to machines.

UNAS is also a run time service that generates a TCP/IP network, independent of application code. The network model, and the run time services for the network, can be generated interactively off-line. Various configurations of the network can also be tested off-line, before application code has been generated, using "burn" statements estimating the load that the application will generate on the computer(s). UNAS allows architectural flexibility, because the same software components, consistent with the DRM, can be associated with many different network and hardware configurations and associated performance constraints. Also, software and physical architecture development can proceed in parallel.

The Application Architecture Model is complete when the Ada Programs, related to the Missions(OSDs), have been mapped to the UNAS Tasks.

DAF in Figure 4, stands for Domain Architecture Framework; the DAF defines the processes which transform the DRM into application specific physical architectures.

4.0 SCAI ARCHITECTURE FRAMEWORK

4.1 ITERATIVE SYSTEM DEVELOPMENT

A key lesson of the SCAI project is that both individual system models, and domain models/architectures, need to be built up iteratively. The connectivity between Ada Components in the DRM is the same as in the AAM; the AAM can be tested off-line by "injecting" simulated messages into the AAM. Therefore, the validity of the DRM can be tested using UNAS, and errors can be uncovered by executing the DRM before coding has proceeded, at the time in the life-cycle when errors cost the least to correct. Also, AAM development can proceed in parallel with Ada Package coding, compressing the time that it takes to develop a system.

4.2 ITERATIVE PROCESS DEVELOPMENT

Megaprogramming mandates the specification and the use of formal processes for the development and maintenance of SCAI models/architectures. Further SCAI experience has also shown the need to iteratively build up these processes and validate them with experience, exactly as with the models/architecture. Several times, as the architecture of the SCAI system has changed, the process has had to change, and vice-versa.

4.3 DEFINITION OF ARCHITECTURE FRAMEWORK

This intimate, synergistic nature between process and architecture must be extended to those other SCAI products: functional specifications, and Ada code, that

have relationships with the architecture, especially because of the iterative nature of the development process. Expressing the relationship between process, architecture, functional specifications, and code is done in the SCAI Architecture Framework. An architecture framework extends domain architecture by:

- including the definition of system specifications and code
- including the processes which build the architecture and system specifications and code

Though not discussed in this document, metrics are also included in the SCAI Architecture Framework to evaluate the quality of the artifacts and architecture.

A SCAI goal was to base the architecture framework on processes with a successful track record. The three major processes that were incorporated into the SCAI Architecture Framework were:

- Booch Object Oriented Analysis for DRM building [Booch93, Booch94]
- Cleanroom Software Development for Requirements Specification, Class translation into Ada Code, and Statistical Testing. [Pal92A]
- TRW Ada Process Model for building the Application Architecture Models and for translating the Service Layer generic services into an application specific product. [Pal92B]

The mapping between the major domain artifacts, and the processes used to build those artifacts, is shown in Figure 5. The mapping is shown to the level of each layer in the architecture. Note that Figure 5 also identifies the fact that SCAI systems have three different architectural views, one per column:

- A Functional View
- A Logical View (both dynamic and static logical relationships)
- A Physical View

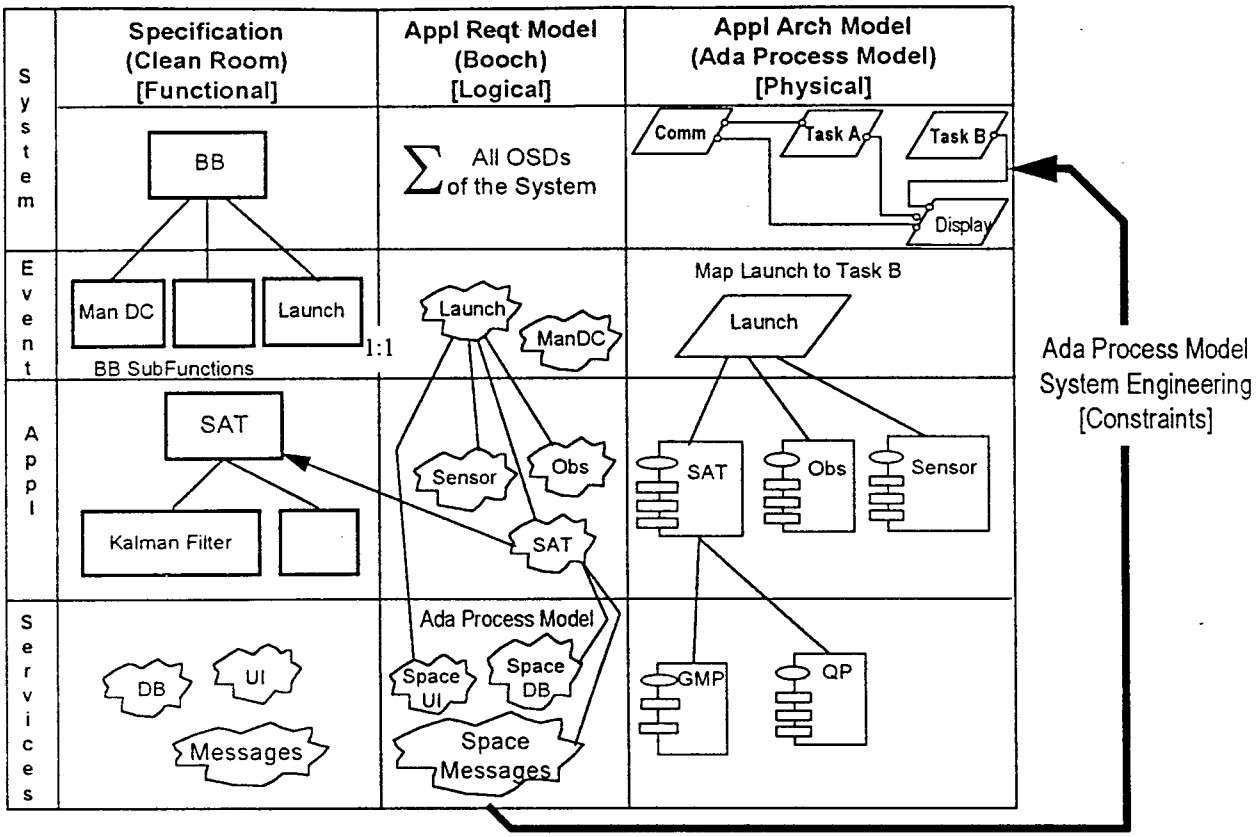


Figure 5. Architecture Framework

System-level artifacts are identified by the top row in Figure 5:

- A system-level functional specification
- A system-level logical model
- A system-level physical mode

Each system level artifact is decomposed into the three layers (Event, Application, and Services) that have been defined previously.

Implicitly, the domain-level logical model (DRM) can also be considered to occupy the system-level logical model spot in the framework; this DRM extends the scope of an individual system to the entire Space Domain, and contains the set of OSD's for the entire Domain.

4.1 Process for Building the Logical Model

System and domain logical models are created, as previously stated, using Booch Domain Analysis. The inputs to this analysis are: Business Process Models,

System Specification Documents, System Design Documents, System ICDs, System Code, and System Operational Concepts Documents.

Roughly speaking, analysis involves proposing strawman generic models, examining existing systems documentation and encoding this information as classes within the models, identifying similarities and differences between the systems, and refining the generic model to accommodate existing systems. Scenarios are used to prove that the generic class models are correct.

4.1.1 Restrictions on SWSC Domain Analysis

At the SWSC, Space Systems that are restructured to conform to the Architecture Framework must still satisfy their original requirements. In general, the same inbound and outbound messages must be processed, and the same display screens must interact with the system users via the same dialogues. Invoked Space Algorithms must have the same accuracy. Therefore, once a good set of Application Objects has been abstracted (grouping and abstracting needed Space Algorithms), the validity of the abstractions is tested by creating object scenario diagrams in the event layer that pass inbound and outbound messages exactly according to current systems functional specifications.

4.1.2 Application Classes into Service Classes

As the Booch Analysis proceeds, common Application Classes are tested to see if they might conform to rules that identify service classes. For instance, if two applications (Space, Missile Warning) both contain alarm processing, then that processing may be isolated in the Services Layer, at the discretion of Domain Management.

4.1.3 Pattern Recognition

The Event layer invokes each Application "Server" that is needed to execute a Mission. Common Services associated with all Missions in the Event Layer (such as error processing) are encapsulated in new class utilities. Also, certain dynamic patterns of use for every Mission have emerged. For instance, whenever a UNAS task is initialized or terminated because a command associated with a Mission has started or terminated, the same type of processing routine is invoked across every mission. To uncover these common patterns of usage, a "pattern recognition" process is used in this layer, which is not object oriented, as well as standard Booch Analysis.

4.1.3 Extending the Domain Incrementally

Every time a new system is incrementally added to the scope of the domain analysis, each class and each mission are examined to see if they can be abstracted with the new domain analysis input data. A variety of options for integrating the new information into the DRM might occur:

- A new class might be added which is independent of existing classes. This does not affect existing Missions.
- An existing class is identified as generalizable, but will not be reabstracted because of the cost of modifying existing systems. A temporary" child class for the new system, tagged as belonging to the new system, will be added to the existing class. The old class will also be tagged as awaiting abstraction.
- An existing class will be modified. New Ada Code will be written for the new class. Any Missions using the class will be retested.

4.1.4 Domain Management

Domain Engineering makes the technical decisions related to extending and abstracting the DRM. An organization called Domain Management makes the scheduling and cost decisions as to whether software should be generalized to reflect these changes and whether the software should be updated and then retested in any or all applications to which it is related.

4.2 PROCESS FOR SYSTEM SPECIFICATION

Cleanroom Software Engineering is used to create the Functional View of SCAI Systems. Cleanroom is a formal method which provides a tightly integrated approach from specification preparation through software development to software certification , bringing engineering rigor and intellectual control to the process [Mills86]. Intellectual control is the ability to clearly understand and describe the problem at hand at the desired level of abstraction. Cleanroom introduces the concept of statistical testing to ensure the developed system meets the users requirements to any level of statistical accuracy. System specifications are coded using Black boxes which explain how stimulus histories are mapped into responses. Black Boxes are then decomposed into State Boxes, which replace stimulus histories with internal state data. The Clear Boxes are then decomposed into Clear Boxes which elaborate the functions that translate stimuli into responses. The process iterates as Clear Boxes are decomposed into new Black Box Subfunctions. This process is called Box Structured Decomposition.

SCAI System Specifications are built using the Cleanroom Software Engineering Process. A system black box (BB) artifact is created, using both the DRM and the same inputs (other than System Design and Code) as used by Domain Engineering. A system BB contains all system stimulus histories (input messages with conditions under

which the message will be accepted), and corresponding system responses, as well as functions which map these stimulus histories into the system responses.

4.3 Process for Functional Decomposition

System BBs are decomposed, via Box Structured Decomposition, into a set of BB subfunctions. Each BB subfunction must map directly into a mission in the DRM Event Layer. Each input message processed by a BB subfunction must be identified in the corresponding mission OSD.

Associated with each stimulus in each BB subfunction, is a probability of the stimulus occurring. The probability is used to evolve a test plan for the subfunction/mission/Ada program that will guarantee that the mission is accurate to any degree of desired accuracy.

Missions are not decomposed using BB decomposition beyond the subfunction level. The decomposition of a subfunction is accomplished using domain analysis, and recorded within the logical model by identifying the classes that occur in the OSD for the mission. BB decomposition cannot be used to identify these classes, because they are generic to the domain, and may imbed methods that do not relate specifically to any one subfunction; in other words, they are reusable across the Space Domain.

Once the Application Classes have been identified (Application Layer, Logical Model), they are translated into Ada Packages using BB decomposition, in order to guarantee the accuracy of the very complicated Space Algorithms.

4.4 Process for Building a Physical Model

As explained previously, the Architecture Model contains UNAS Tasks imbedded with Ada Programs representing Missions in the Event Layer. Application Layer Classes identified in OSDs are included (in the form of Ada Packages) within these Mission Ada Programs. The system level physical model is represented by the upper right box in Figure 5.

The Architecture Model is built using methodology defined in the Ada Process Model, a systems engineering process specific to Command and Control Systems. The TRW Ada Process Model [Royce89,90a,b;PAL92] is an iterative, demonstration-based, code generator-based, development technique particular to Command and Control Systems

Once basic system missions have been defined in the logical model, the BB subfunctions have been defined in the functional specification, and performance constraints have been identified for the system, Ada Process Model (APM) iteratively builds proposed physical architectures, identifying flaws in each proposed architecture, and then correcting those errors.

APM addresses typical system engineering concerns by performing "trade studies" that balance cost against hardware and storage resources.

The generic Common Services identified in the Service Layer of the Functional column of Figure 5, must be made system-specific. For instance, each display format for the SCAI Application contains a command line, which must be parsed. Therefore, a parsing service is added to display services for the SCAI Application. All system-specific services for a single application reside in the Service Layer under the Logical Column. An encapsulating class is defined for each particularized service. The encapsulated classes are used to reference these classes in the Logical Model Event and Application Layers.

4.5 Related Process Paper

Note that the process for integrating the Logical, Physical, and Functional development processes on the SCAI Project will be discussed in the paper "Product Line Software Development" by David J. Bristow also under the Architecture Track at STC.

5.0 Lessons

The following lessons in Domain Engineering relate to both the creation of the Architecture Framework, and its integrated processes, and to the execution of those processes on the SCAI project. The lessons have been abstracted so that they have broader scope than just to the SCAI project, or just to the Space Domain. These lessons, among other SCAI Project lessons, will be formally delivered in "AF/STARS Demonstration Project Experience Report Version 2.0," Contract No. F19628-93-C-0129, CDRL Sequence A011-002D.

5.1 Lesson: The DE Process Must Allow for Iterative Domain Knowledge Acquisition.

Product Line Software Development implies two software life cycles: one life cycle developing generalized domain products and a parallel life cycle developing individual applications which are constructed from the domain life cycle products. Obviously, domain products must exist prior to their use on the application. Problems associated with the prior construction of all domain components are:

- Great upfront DE costs not associated with developing any product. (As DoD shrinks, these kind of costs are increasingly difficult to justify).
- Generalized Models and generalized components can only be validated through their use on real systems. Even within a single system, a reusable class must be validated in each of the contexts in which it is

used. Monolithic "water-fall" systems development has largely been discredited vis-à-vis more iterative approaches to modeling and systems development.

- A large domain, such as Command and Control, may contain very many systems. In spite of the fact that all these systems have much in common, it is unlikely that DE can be initially accomplished with the scope of every one of these systems, before the need to deliver the first rearchitected system occurs.

For the above reasons, the DE process must allow for the iterative acquisition of domain knowledge. The SCAI Architecture Framework was constructed from processes which are inherently iterative. Therefore, the overall process is iterative. As new systems are analyzed, and the scope of the DRM is extended, more domain missions are identified, new classes are created, and existing classes are generalized. Newly generalized classes are reinserted into existing missions, and the missions are retested.

The cost and time to perform the Domain Analysis (DA) of the Space Domain, on the SCAI Project, was underestimated and the DA analysis is not currently complete. However, the ability to continue to build generic software has not been halted, because the DA process is iterative.

5.2 LESSON: INTERPRET THE STARS "TWO-LIFECYCLE MODEL" AS A SINGLE INTEGRATED DE/AE PROCESS.

The SCAI team has found it necessary to interpret the STARS "Two-Lifecycle Model" to show the very close interaction between Domain Engineering and Application Engineering, in order to constantly validate the domain models against real applications in the domain. In the original STARS model, Figure 6, the parallel Domain Engineering Lifecycle creates a domain design and domain implementation. These generalized artifacts must be kept consistent with application specific systems designs and systems implementations in the Application Life Cycle.

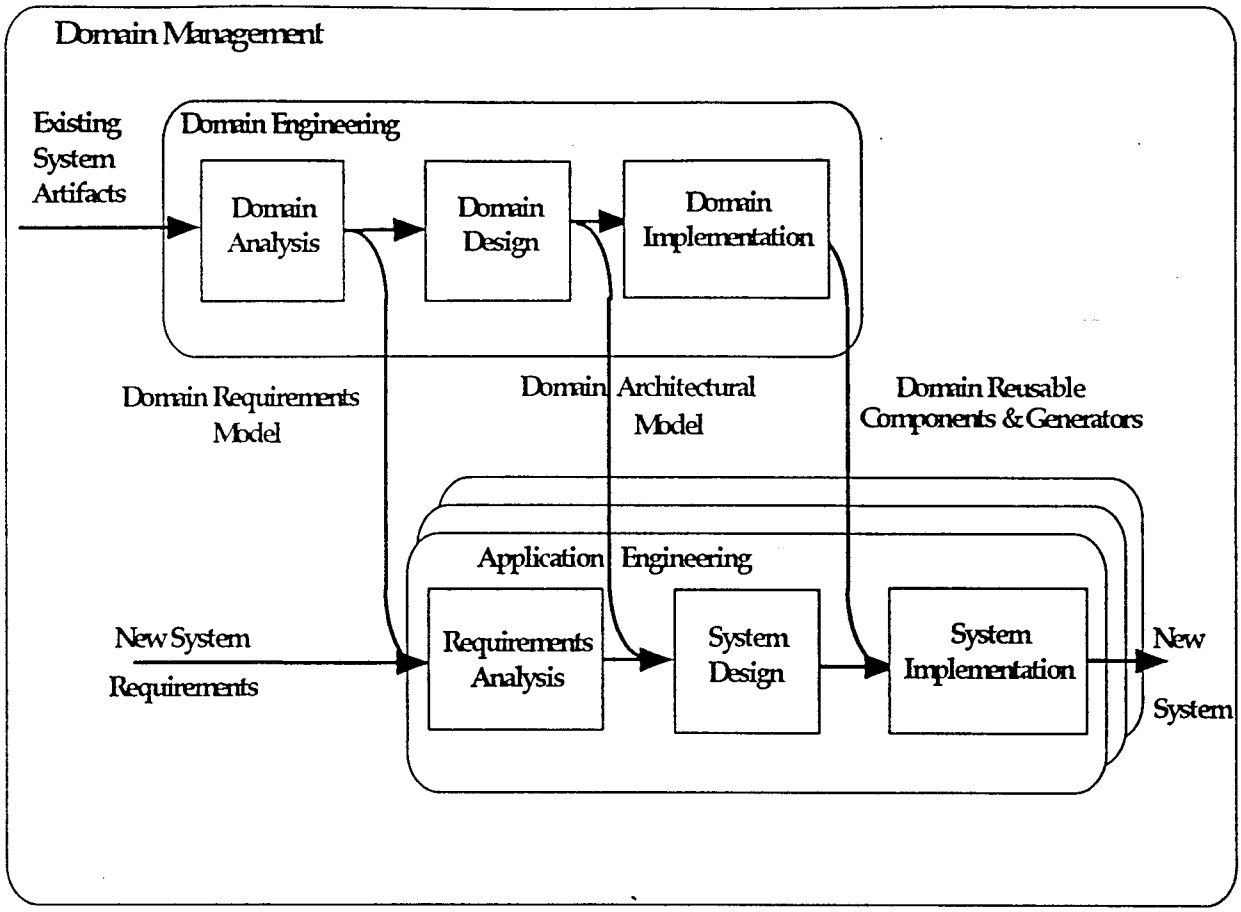


Figure 6. STARS Two Life-Cycle Model

To avoid serious configuration management problems, SCAI decided to maintain a single logical model, the DRM, used by both the Domain and Application Lifecycles, Figure 7. Applications would add detail to the model; The Domain Organization would add new classes when extending the scope of the domain, and potentially generalize existing classes within the previous scope of the domain. All system specifications and system components would point to this single DRM. Component generation, both generic to more than one system, and specific to a single system, would be assigned to the development Lifecycle of an existing Application. Testing for the generalized components would then have to occur in each of the applications to which the component applied. Test cases and infrastructure are maintained with each application, so retesting within an existing application should be routine. Again, the cost decision to actually reintegrate the generalized component within an existing application is a decision of the Domain Management organization

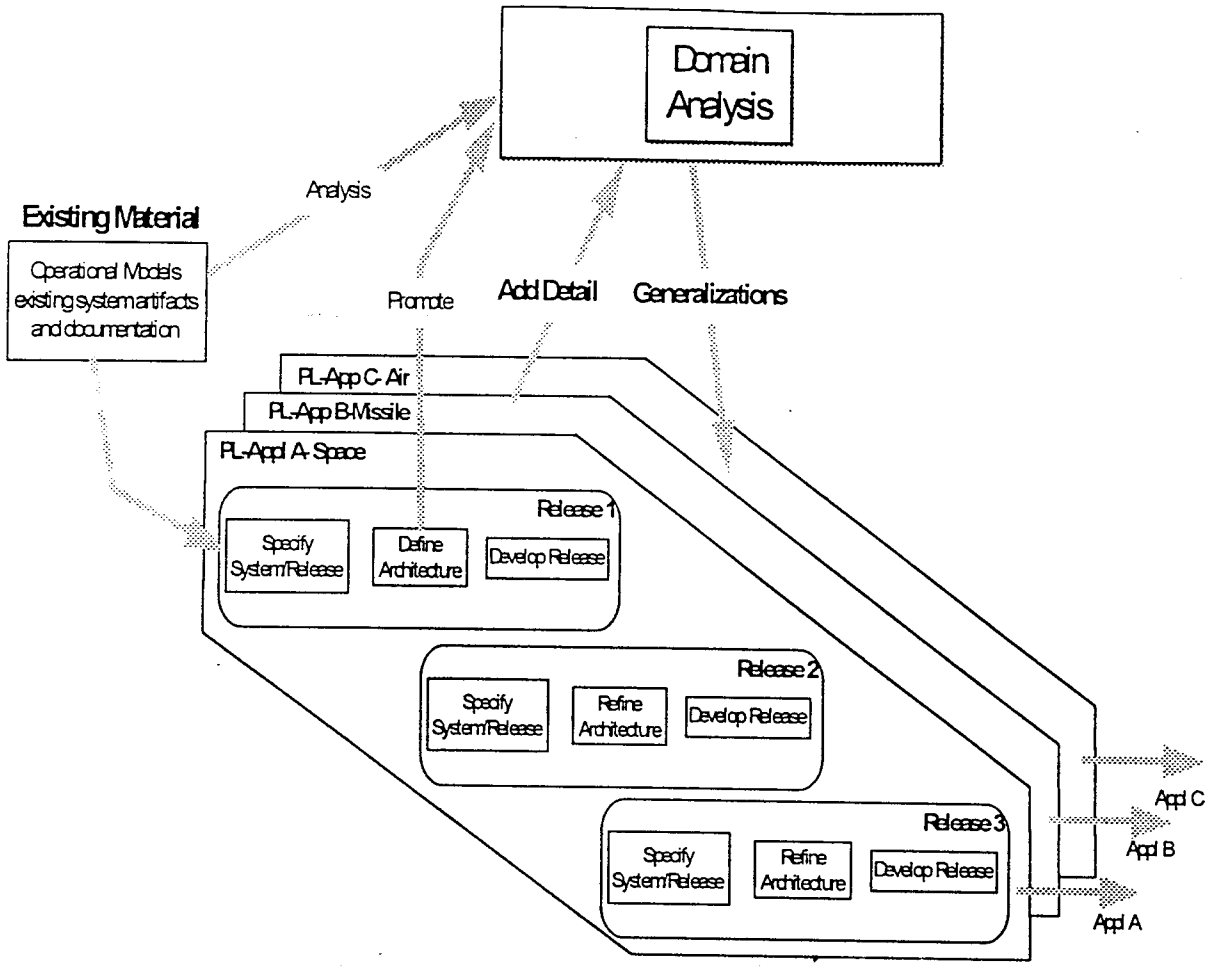


Figure 7. Modified Two Life-Cycle Model

5.3 LESSON: THE DOMAIN ANALYSIS PROCESS SHOULD BE APPLICABLE TO INDIVIDUAL APPLICATIONS.

Domain Analysis (DA) typically looks at multiple systems with the intent of discovering the similarities and the differences between those systems. The overlap of DA with Systems Analysis should be in identifying common abstractions across any scope of analysis; common abstractions will result in common code components. If the DA process is well-understood it should be applicable to a single system. When applied to a non-perfect, single system, the result of DA should be a simpler, well-structured, single system. Thus, even if an organization is initially unwilling to invest in DA and generalized component development across the whole extent of the domain, they may be willing to apply domain analysis to a single system, and still be able to judge the economic and technical value of the DA process. That the DA process is not affected by the scope of its application, is essential to doing incremental domain analysis as specified in Paragraph 4.1. This does not change the central purpose of domain analysis from identifying commonalties across multiple systems.

5.4 LESSON: USE VALIDATED TECHNOLOGIES FOR DOMAIN ENGINEERING

SCAI believes that Domain Specific Software Reuse may require dramatic cultural change, but not sophisticated new technologies. It has been shown previously that there is much overlap between Systems Engineering and Domain Engineering. Domain Specific Reuse should be implemented using technologies that are intuitively easy to understand, and have widespread use. Then, the cultural shock associated with megaprogramming will not be reinforced by a technological learning curve.

For example, existing Object Oriented methods (for instance, Booch Analysis) are sufficient for recording the results of DA. Superclasses, class categories, and polymorphism intrinsically identify high level abstractions/system commonalties. Sub-classes identify system differences. Classes provide "real world" abstractions that do not require programming expertise to understand. At a high level of abstraction, a Satellite Class is easier to understand than a set of functionally decomposed orbital calculation algorithms. Satellite relationships to groundstations and orbits encoded in a class diagram provide much more information that a simple hierarchical decomposition.

Originally, SCAI Domain Analysis was going to be performed using the methodology of Ruben Prieto-Diaz [Diaz91-2]. Even though the methodology was very promising, SCAI was unwilling to use it because it did not have the track record of the more prosaic Object Oriented Methodology. However, key aspects of the methodology were incorporated into Booch, such as the bi-directional mapping of low level system abstractions to a high level architecture, and revisions.

SCAI Configuration management of domain and application artifacts has been accomplished using standard tools such as Rational CMVC, and does not require the use of a Software Reuse Library Mechanism. The Software Reuse Community originally thought that the key to software reuse was making components available in a Reuse Library Mechanism having sophisticated search mechanisms. However, SCAI recognizes that the DRM is the key to understanding and finding all domain software components.

5.5 LESSON: DOMAIN ANALYSIS SHOULD EXAMINE THE RELATIONSHIP OF PEOPLE/ORGANIZATIONS TO SYSTEMS

A restriction to SCAI Domain Analysis is that SCAI rearchitected systems must retain their existing external interfaces. SCAI DA stops at system boundaries. This restriction seriously degrades the ability of the Domain Engineering Organization to improve existing systems based on new technology. For example, in the Space Domain, several systems involve the Orbit Analyst in a complicated dialogue across multiple display screens in order to determine the orbit of a satellite that requires non-routine calculations. The original reason for creating the dialogue, was that these non-routine calculations require a great deal of compute power, and the human (Orbit

Analyst) was part of the loop to determine the situations in when the superior accuracy of the nonroutine calculations was critical. As hardware becomes faster, the nonroutine calculations can be performed every time, and the dialogue with the human can disappear altogether.

Human roles and computer roles in the overall "business process" for an organization need to be included in an accurate Domain Analysis. The SWSC currently does Business Process Modeling, but this activity is not integrated with SCAI Domain Engineering.

5.6 Lesson: A Domain Model Must Contain Behavioral Abstractions

SCAI requires that application functional specifications be kept consistent with the DRM. If this consistency is maintained, then SCAI systems will satisfy their functional requirements. Functional specifications show how system input messages are mapped to system output messages. It must be shown that when the same messages enter the DRM that a path exists through the DRM such that the same output messages will be generated as in the functional specification. This need to trace messages through the DRM requires that behavioral abstractions be included in the DRM.

OSDs define complete paths between objects in the DRM, when associated with State Transition Diagrams and Ada PDL to specify the logical conditions under which messages may flow. State Transition Diagrams and Ada PDL may also have to be associated with individual objects that have complicated behavior, to identify the logical conditions under which messages will be passed out of the object.

When all paths through the DRM are maintained as part of the DRM, then it can be considered "executable" and consistent with its requirements.

5.7 Lesson: Domain Analysis Should Produce the Simplest Reusable System.

Behavioral abstraction is also needed to evaluate the complexity of the DRM. Static Domain Analysis of the functionality that a domain must contain is not enough to ensure that reusable components can be combined into well-architected, simple to understand systems. DA should identify components that will result in the *simplest possible* reusable system within the context of the domain.

SCAI DA compares the complexity of the behavior encapsulated within classes (class state transition diagrams) relative to the complexity of the interfaces defined between those classes. Interfaces between classes are represented by OSDs as well as the State Transition Diagrams associated with each OSD. If the OSD is very complicated, requiring interfaces to many objects, then the mission represented by the OSD is tightly coupled. In order to loosely couple the OSD, the class definitions will be

reformulated. Standard Object Oriented Metrics are used to evaluate the complexity of the model.

5.8 Lesson: Processes Are Replaceable in an Architecture Framework

At least three different domain analysis processes have been used to abstract a set of common services for Command and Control Systems. Ruben Prieto-Diaz applied his Domain Analysis Process Model at Contel [Diaz91-2]. TRW Corporation applied an internal DA method to identify the RICC Services used by SCAI. Loral Corporation also used an internal method to abstract common services for its CCS-2000 Domain Specific Architecture. In all three cases, the same set of services were identified.

Booch Analysis was chosen to elaborate the logical architecture. However, other Object Oriented methods were examined during the SCAI preparation phase (Rumbaugh, Shlaer-Mellor, and others), and, from the method selection analysis, other methods also would have been good choices for decomposing the logical architecture.

Cleanroom Software Engineering was chosen to decompose the functional view of SCAI systems, but a variety of structured analysis techniques could have been substituted for Cleanroom.

The point is, that once an architecture framework has been defined for a domain, a variety of processes can be used to develop the artifacts within different layers of the Architecture Framework. The reason for choosing a particular process may simply be that personnel are familiar with that process.

5.9 Lesson: Layer the Domain Architecture to Insulate the Domain from Technological Change

When technological changes mandate system changes, systems should not have to be redeveloped from scratch. Big DoD systems are currently migrating from large mainframe computers to networked workstations. The cost of rearchitecting these Command and Control systems is proving exorbitant. The next such technology transition might be toward massively parallel computers.

Typically, a DRM would not contain a service layer, because activities such as querying a relational data base, are not normally part of an abstract, problem-state model. However, SCAI recognized that identifying common services, and placing them in a layer of the DRM not only simplified the domain analysis process, but facilitated the process of "encapsulating" these services. Once classes have been defined to encapsulate these services, the cost of replacing these services with equivalent services can be readily estimated, because all references to the services are graphically clear in the Class Diagram of the DRM.

An obvious future service replacement to the SCAI architecture, would be to replace the current relational data base with an object oriented data base, or to replace UNAS with a CORBA compliant networking service.

Future SCAI Domain Engineering plans call for abstracting the current DRM Event Layer into two layers: the current Event Layer plus a User Interface Layer. The User Interface Layer would totally isolate dialogues between a system and its users, to allow simple replacement of the Display service, and to identify common dialogues and simplify the way users interact with SCAI systems.

5.10 Lesson: Product-Line Organizations Should Be Constructed According to the Architecture Framework

Contractors and Air Force personnel on the SCAI project have been co-located by subcontractor, and not by the process in the Architecture Framework that they are performing. Communication and synergy are lost because it is sometimes difficult for personnel executing the same process to "feed" off each others expertise.

Each Architecture Framework process requires a different set of skills and knowledge to perform. For example, executing the Ada Process Model requires skills in network architecture and systems engineering, while building a mission in the Event Layer of the DRM requires an understanding of space operations.

Product-line personnel should be organized into functional organizations based on the architecture and be co-located to facilitate information exchange, which would also avoid duplication of the Software Engineering Environment (SEE) facilities.

Organizational structures that parallel the architecture framework, enhance the communications needed to maximize reuse opportunities and deliver on the promised Megaprogramming productivity increases. This organizational approach also avoids duplication of effort.

The "other side of this coin" is that an architecture should be originally developed for *organizational* reasons as well as technical reasons. The SCAI Mission layer of the DRM was created partially to ensure that space operators could understand the domain without being forced to understand the mathematics of the Application layer and the Software Engineering of the Services Layer.

5.11 Lesson: Validation Data Is Associated with Missions, Not Reusable Components

Cleanroom Software Engineering uses probabilities associated with the likelihood of system stimuli occurring, and the "amount" of testing performed, to determine the statistical likelihood that programs will fail. SCAI Stimuli are associated with Missions. Therefore, validation data (test results, test cases, test data, etc.) relate to Missions; no statistical accuracy is associated with any specific reusable component.

Reusable components are *not* unit tested. Therefore, no contention is made, or can be made, about the validity of a reusable component outside the context of its use.

This is just another specific example of Domain Specific Reuse, where no contention is made that a domain component is reusable outside of the domain context identified by the domain model.

5.12 Lesson: Both Functional and Object Oriented Mind Sets are Required to Build a Space Architecture Framework

Space systems requirements are functionally stated; the technology to develop software and test it to a specified degree of statistical accuracy is functional. Most COTS and GOTS reusable software is functional. Therefore, for reuse in the present day, it is necessary to provide a functional view of the Architecture.

However, Object Oriented Modeling, at least according the SCAI philosophy, is the best way of decomposing a domain so that it can be *understood,* and is *reusable.* However, one lack in Object Oriented Modeling, according to some technologists, is that it is an informal modeling technique. Domains are not decomposed into precise, verifiable units. Therefore, there is a concerted effort to combine formal methods with Object Oriented Modeling, in order to add this precision. Cleanroom Software Engineering imbeds semi-formal methods for software generation, and therefore, remedies Object Oriented Modeling flaws.

5.13 Lesson: Megaprogramming Projects Still Need to Focus on Basic Engineering and Project Management Disciplines.

New technology is always viewed as a "silver bullet." However, no technology obviates the need for bright system engineering and management talent, and people who understand the domain of the product line. In fact, as the technologies become more sophisticated, the need to have talented people actually increases.

Although Megaprogramming brings to the table multiple concepts to improve the production of software systems, it does *not* replace some of the basic engineering and project management disciplines.

Megaprogramming extends the scope of software development, requiring that more and more people work together (though eliminating redundant personnel). Establishing an efficient organization with cooperating domain and multiple application organizations, all building software incrementally and cooperatively, is a formidable management challenge.

Regardless of how well defined domain components are, inserting those components into an efficient network architecture still requires sophisticated systems engineering talent.

5.14 LESSON: DOMAIN ANALYSIS NEEDS TO BE SUPPORTED BY A PRODUCT LINE ORGANIZATION.

The SCAI project conducted domain analysis on the Space Domain which included the SPADOC 4C data base. SPADOC 4C is a Space System, partially deployed in Cheyenne Mountain, that was analyzed via SCAI domain analysis. A number of redundancies were uncovered in this data base, which presumably was constructed incrementally over the life of the SPADOC system. Due to the stovepipe organizational structure and resulting differences in priorities, it has been difficult to cooperate in this area (which could benefit both SCAI and SPADOC). We believe that organizational structuring based on the product-line approach would enhance the SWCS's ability to take advantage of Domain Analysis, increase the focus on commonalties between systems, avoid reporting boundaries and facilitate information sharing between current systems and new ones.

From the above small experience, deploying SCAI Megaprogramming technologies, will be far harder than inventing the SCAI Megaprogramming technologies in the first place.

5.15 LESSON: PATTERN RECOGNITION WORKED FOR THE SCAI PROJECT.

It was presumed that reuse would occur in the Application Layer of the DRM as multiple missions in the Event Layer used the same Application Layer Classes. In fact, as missions were developed, not only was it found that as each mission was elaborated, the same common services were required to support the mission, but also that the services were applied in the same order. Thus, a whole set of reusable services were created, and were encapsulated in Class Categories called Event Control and Event Common. This type of discovery of reuse opportunities (through order of class invocation) is an example of a new theory of architectural reuse referred to in the literature as "Pattern Recognition."

5.16 LESSON: CRITICAL PATHS OF THE PHYSICAL ARCHITECTURE MUST BE ITERATIVELY TESTED.

The Ada Process Model, incorporated into the AE/DE process, demands that a system, including its architecture, be iteratively built and tested. For example, the original SCAI Domain Architecture did not distribute the human-machine interface module to each workstation, creating too much message traffic on the network. This overload condition was identified because UNAS allowed for architectural testing before any significant application code had been written. This enabled the architect to correct the architecture without impact to the schedule. This is a specific example of the general need to iteratively develop and test the physical architecture.

5.17 Lesson: Ada Code Generators created significant amounts of reusable code.

Ada Code Generation, such as that provided by the Reusable Integrated Command Center (RICC) Tool set, has greatly decreased SCAI code development time. RICC infrastructure provided reuse across the entire space domain including user interface generation, message parsing/assembly, network control, and data base support. RICC was used to help elaborate user interfaces rapidly and gain customer acceptance. RICC provided 52% of the SCAI Release 1 code (19% reused, 33% generated).

5.19 Lesson: Megaprogrammers Must Understand Process Definition

Megaprogrammers not only need skills in software development, but also in process definition and improvement. Process definition must be recorded by those who will use it, so that they will enthusiastically accept the process, identify inefficiencies in it, and identify needed process improvements. On the SCAI Project, when software development organizations did not record the processes they used, they felt free to ignore those processes.

6.0 Summary

The SCAI Project focuses on how standard, proven software engineering technologies can be used for Domain Engineering, not on the need for unique new technologies.

SCAI emphasizes that Software Process and Software Architecture are intimately related, using the Architecture Framework to show the relationships.

SCAI recasts Domain Engineering as an *incremental* process that is *tightly coupled* with Application Engineering. Incremental applies to both the elaboration of the architecture and the elaboration of the process that builds the architecture.

Finally, SCAI emphasizes that Product Line Organizations are required to implement Megaprogramming and that these organizations should conform to the architecture used to subdivide the Domain.

REFERENCES

[Booch94] Booch G., ***Object-Oriented Analysis and Design***, 2nd Edition, Benjamin Cummings, 1994

[Bristow93] Bristow D.J. (Lt.Col. CF), ***Space Command and Control Architectural Infrastructure (SCAI) Air Force/STARS Demonstration Project Management Plan. Appendix B RICC***, Version 2.0, 15 Oct 93

[DemExp95] AF, Loral, ***AF/STARS Demonstration Project Experience Report, Version 2.0 (Draft)***, CDRL A011-002D, December 1994 (currently under Government review)

[IBM93a] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 1 Cleanroom Engineering Process Introduction and Overview***, 31 Jul 93, STARS-05507-001A

[IBM93b] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 2 Organization and Project Formation in the Cleanroom Environment***, 31 Jul 93, STARS-05507-001B

[IBM93c] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 3 Project Execution in the Cleanroom Environment***, 31 Jul 93, STARS-05507-001C

[IBM93d] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Specification Team Practices***, 31 Jul 93, STARS-05507-001D

[IBM93e] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Development Team Practices***, 31 Jul 93, STARS-05507-001E

[IBM93f] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Certification Team Practices***, 31 Jul 93, STARS-05507-001F

[Mills86] Mills H.D., Linger R.C., Hevner A.R., ***Principles of Information Systems Analysis and Design***, Academic Press, 1986

[PAL92a] Arnold P.G. ***Cleanroom Engineering Process***, STARS/SEI Process Asset Library, 1 Oct 92

[PAL92b] Pixton J. Maymir-Ducharme F. ***The TRW Ada Process Model***, STARS/SEI Process Asset Library, 16 Oct 92

[Diaz91-1] Prieto-Diaz, Ruben, Reuse Library Process Model, Loral FS CDRL # 0341-002, 07-26-91

{Diaz91-2] Prieto-Diaz Ruben, Arrango Guillermo, ***Domain Analysis and Software Systems Modeling***, IEEE computer Society Press, 1991

[Royce89] Royce W. ***Ada Process Model***, TRW SEDD Guidebook, 5 Oct 89, SEDD SGB-01-10-89-A

[Royce90a] Royce W., ***TRW's Ada Process Model for Incremental Development of Large Software Systems***, TRW Technologies Series, Jan 1990, TRW-TS-90-01

[Royce90b] Royce W., ***TRW's Ada Process Model for Incremental Development of Large Software Systems***, Proceedings of the 12th International Conference on Software Engineering, Nice France, 26-30 March 1990

Appendix J

References

1. "Application Portability Profile (APP) The U.S. Government's Open System Environment Profile OSE/1", Version 1.0, Apr 91, National Institute of Standards and Technology (NIST).
2. Technical Reference Model for Information Management", Version 1.2, 15 May 1992, Defense Information Systems Agency/Center for Information Management (DISA/CIM).
3. "Infrastructure Implementation Assessment Service Engineering Report", 19 March 1993, TRW.
4. "Display Builder Users Manual", 11 January 1993, TRW.
5. "HMI User's Manual", 11 January 1993, TRW.
6. "Generic Message Parser User's Manual", 15 April 1993, TRW.
7. "Message Tool User's Manual", 13 April 1993, TRW.
8. "Query Builder User's Manual", 8 February 1993, TRW.
9. "Query Processor User's Manual", 13 April 1993, TRW.
10. [Conner 82] Conner, Daryl R. and Robert W. Patterson; Building Commitment to Organizational Change, Training and Development Journal, April 82.
11. [Fowler 88] Fowler, Priscilla and Stanley M. Przybylinski; Workshop Overview, Transferring Software Tools and Technology, IEEE Computer Society Press, November 88.
12. [Curtis 88] Curtis, Bill, Herb Krasner, and Neil Iscoe; A Field Study of the Software Design Process for Large Systems, November 88.
13. [Moore 92] "Crossing the Chasm", Jeoffrey A. Moore, Harper Business Inc., 1992.
14. [Dart 90] Dart, S.,"Spectrum of Functionality in Configuration Management Systems", Technical Report SEI-90-TR-85, Software Engineering Institute, December 1990.
15. [Contract] "STARS Follow-on Contract", F19628-93-C-0129, ESC USAF, Hanscom, August 1993.
16. [DID30255] "Informal Technical Data", USAF, May 1977.
17. [Proposal] "IBM Proposal for STARS Follow-on Contract", IBM/FSC Gaithersburg, April 1993.
18. [IBM5052] "Demonstration Project Baseline Report", IBM STARS CDRL No. 5052, IBM, April 1993.
19. [IBM5200] "SEE Integration Plan", IBM STARS CDRL No. 5200, IBM, June 1993.
20. [NGCRRM] "Reference Model for Project Support Environments", Version 2.0, Navy Next Generation Computer Resources/ Project Support Environment Standards Working Group, September 1993.
21. [NISTRM] "Reference Model for Frameworks of Software Engineering Environments", Special Publication 500-201, NIST, December 1991.
22. [NISTAPP] "Application Portability Profile", Special Publication 500-187 (revised), NIST, May 1993.
23. [POSIX] "Draft Guide to the POSIX Open Systems Environments", P1003.0, IEEE, June 1992.
24. [IBMA010] "SCAI SEE Description", IBM STARS CDRL No. A010, IBM, October 1993.
25. [BROWN92} Brown, A. W., Earl A. N., and J. A. McDermid, "Software Engineering Environments: Automated Support for Software Engineering", McGraw-Hill Book Co., UK, 1992.

26. [ToolTalk94] "Common Desktop Environment: Getting Started Using ToolTalk Messaging", Sun Microsystems, In., Mountain View, CA, 1994.

27. [RandallA95} Randall, Richard, Robert Ekman, Scott Kent, Gary Turner, Integrating a SEE for Megaprogramming, February 1995.

28. [RandallB95} Randall, Richard, Will Ett, Using Process to Integrate Software Engineering Environments, February 1995.

29. [Bulat95} Bulat, Brian, SWSC Domain Engineering Experience, February 1995.